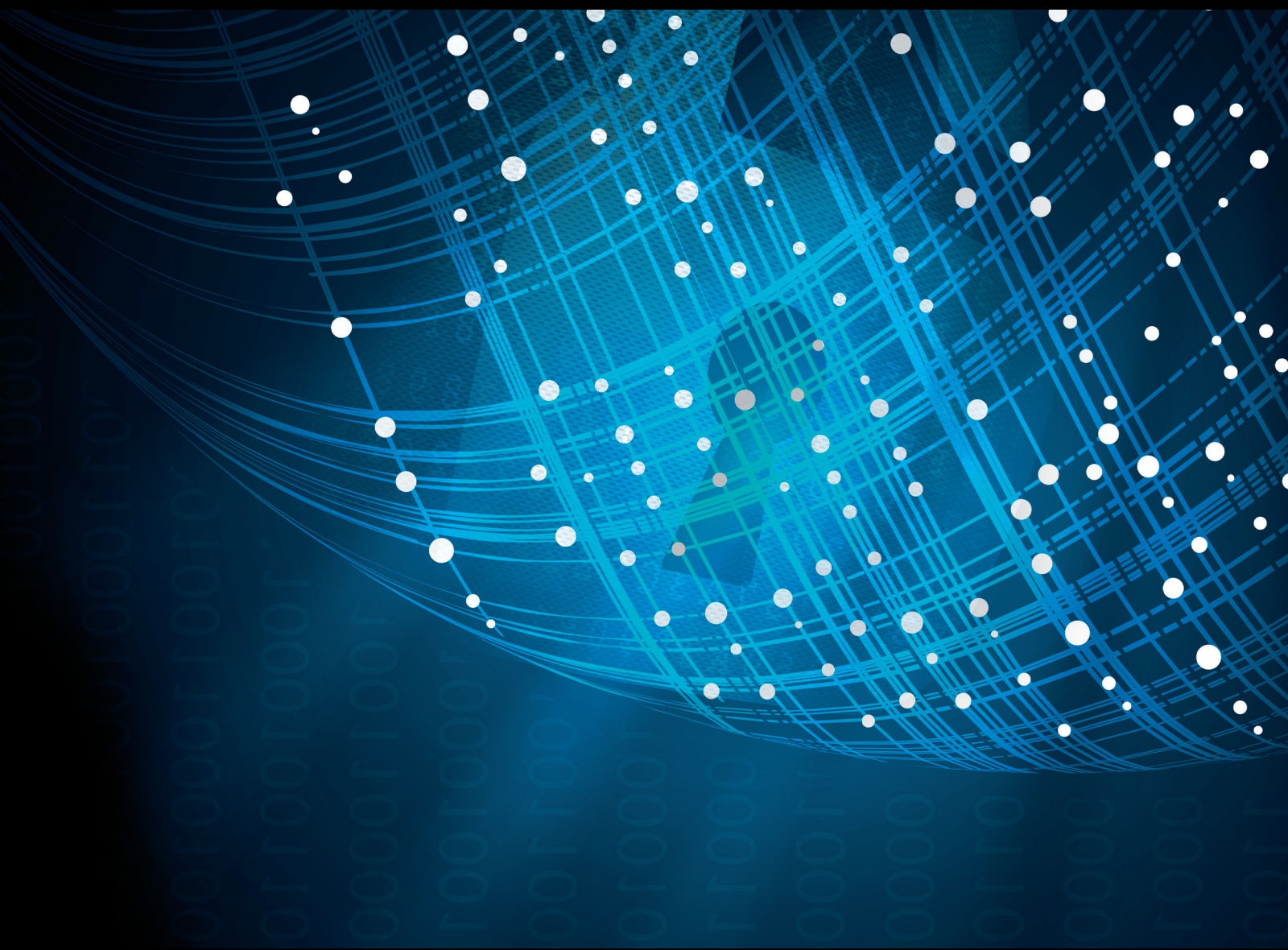
Security and Communication Networks

Advances in Deep Learning Methods for Cyber Attack Recognition, Prediction, and Mitigation

Lead Guest Editor: Robertas Damaševičius
Guest Editors: Nureni Ayofe Azeez and Lalit Garg



Advances in Deep Learning Methods for Cyber Attack Recognition, Prediction, and Mitigation

Security and Communication Networks

Advances in Deep Learning Methods for Cyber Attack Recognition, Prediction, and Mitigation

Lead Guest Editor: Robertas Damaševičius
Guest Editors: Nureni Ayofe Azeez and Lalit Garg

Copyright © 2023 Hindawi Limited. All rights reserved.

This is a special issue published in “Security and Communication Networks.” All articles are open access articles distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Chief Editor

Roberto Di Pietro, Saudi Arabia

Associate Editors

Jiankun Hu, Australia
Emanuele Maiorana, Italy
David Megias, Spain
Zheng Yan, China

Academic Editors

Saed Saleh Al Rabaee, United Arab Emirates
Shadab Alam, Saudi Arabia
Goutham Reddy Alavalapati, USA
Jehad Ali, Republic of Korea
Jehad Ali, Saint Vincent and the Grenadines
Benjamin Aziz, United Kingdom
Taimur Bakhshi, United Kingdom
Spiridon Bakiras, Qatar
Musa Balta, Turkey
Jin Wook Byun, Republic of Korea
Bruno Carpentieri, Italy
Luigi Catuogno, Italy
Ricardo Chaves, Portugal
Chien-Ming Chen, China
Tom Chen, United Kingdom
Stelvio Cimato, Italy
Vincenzo Conti, Italy
Luigi Coppolino, Italy
Salvatore D'Antonio, Italy
Juhriyansyah Dalle, Indonesia
Alfredo De Santis, Italy
Angel M. Del Rey, Spain
Roberto Di Pietro, France
Wenxiu Ding, China
Nicola Dragoni, Denmark
Wei Feng, China
Carmen Fernandez-Gago, Spain
AnMin Fu, China
Clemente Galdi, Italy
Dimitrios Geneiatakis, Italy
Muhammad A. Gondal, Oman
Francesco Gringoli, Italy
Biao Han, China
Jinguang Han, China
Khizar Hayat, Oman
Azeem Irshad, Pakistan
M.A. Jabbar, India
Minho Jo, Republic of Korea
Arijit Karati, Taiwan
ASM Kayes, Australia
Farrukh Aslam Khan, Saudi Arabia
Fazlullah Khan, Pakistan
Kiseon Kim, Republic of Korea
Mehmet Zeki Konyar, Turkey
Sanjeev Kumar, USA
Hyun Kwon, Republic of Korea
Maryline Laurent, France
Jegatha Deborah Lazarus, India
Huaizhi Li, USA
Jiguo Li, China
Xueqin Liang , Finland
Zhe Liu, Canada
Guangchi Liu, USA
Flavio Lombardi, Italy
Yang Lu, China
Vincente Martin, Spain
Weizhi Meng, Denmark
Andrea Michienzi, Italy
Laura Mongioi, Italy
Raul Monroy, Mexico
Naghmeh Moradpoor, United Kingdom
Leonardo Mostarda, Italy
Mohamed Nassar, Lebanon
Qiang Ni, United Kingdom
Mahmood Niazi, Saudi Arabia
Vincent O. Nyangaresi, Kenya
Lu Ou, China
Hyun-A Park, Republic of Korea
A. Peinado, Spain
Gerardo Pelosi, Italy
Gregorio Martinez Perez, Spain
Pedro Peris-Lopez, Spain
Carla Ràfols, Germany
Francesco Regazzoni, Switzerland
Abdalhossein Rezai, Iran
Helena Rifà-Pous, Spain
Arun Kumar Sangaiah, India
Nadeem Sarwar, Pakistan
Neetesh Saxena, United Kingdom
Savio Sciancalepore, The Netherlands

De Rosal Ignatius Moses Setiadi, Indonesia
Wenbo Shi, China
Ghanshyam Singh, South Africa
Vasco Soares, Portugal
Salvatore Sorce, Italy
Abdulhamit Subasi, Saudi Arabia
Zhiyuan Tan, United Kingdom
Keke Tang, China
Je Sen Teh, Australia
Bohui Wang, China
Guojun Wang, China
Jinwei Wang, China
Qichun Wang, China
Hu Xiong, China
Chang Xu, China
Xuehu Yan, China
Anjia Yang, China
Jiachen Yang, China
Yu Yao, China
Yinghui Ye, China
Kuo-Hui Yeh, Taiwan
Yong Yu, China
Xiaohui Yuan, USA
Sherali Zeadally, USA
Leo Y. Zhang, Australia
Tao Zhang, China
Youwen Zhu, China
Zhengyu Zhu, China

Contents

LogPal: A Generic Anomaly Detection Scheme of Heterogeneous Logs for Network Systems
Lei Sun and Xiaolong Xu
Research Article (12 pages), Article ID 2803139, Volume 2023 (2023)

Internet-of-Things-Based Suspicious Activity Recognition Using Multimodalities of Computer Vision for Smart City Security
Amjad Rehman, Tanzila Saba, Muhammad Zeeshan Khan, Robertas Damaševičius, and Saeed Ali Bahaj
Research Article (12 pages), Article ID 8383461, Volume 2022 (2022)

E-minBatch GraphSAGE: An Industrial Internet Attack Detection Model
Jin Lan, Jia Z. Lu, Guo G. Wan, Yuan Y. Wang, Chen Y. Huang, Shi B. Zhang, Yu Y. Huang, and Jin N. Ma
Research Article (12 pages), Article ID 5363764, Volume 2022 (2022)

Light Weighted CNN Model to Detect DDoS Attack over Distributed Scenario
Harish Kumar, Yassine Aoudni, Geovanny Genaro Reivan Ortiz, Latika Jindal, Shahajan Miah, and Rohit Tripathi
Research Article (10 pages), Article ID 7585457, Volume 2022 (2022)

Memory-Augmented Insider Threat Detection with Temporal-Spatial Fusion
Dongyang Li, Lin Yang, Hongguang Zhang, Xiaolei Wang, and Linru Ma
Research Article (19 pages), Article ID 6418420, Volume 2022 (2022)

A Deep Learning Method for Android Application Classification Using Semantic Features
Zhiqiang Wang, Gefei Li, Zihan Zhuo, Xiaorui Ren, Yuheng Lin, and Jieming Gu
Research Article (16 pages), Article ID 1289175, Volume 2022 (2022)

Hindawi
Security and Communication Networks
Volume 2023, Article ID 2803139, 12 pages
https://doi.org/10.1155/2023/2803139

Research Article

LogPal: A Generic Anomaly Detection Scheme of Heterogeneous Logs for Network Systems

Lei Sun [1] **and Xiaolong Xu** [1,2]

[1]*Jiangsu Key Laboratory of Big Data Security & Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*
[2]*School of Computer Science, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*

Correspondence should be addressed to Xiaolong Xu; xuxl@njupt.edu.cn

Received 5 February 2022; Revised 26 September 2022; Accepted 12 October 2022; Published 11 April 2023

Academic Editor: Lalit Garg

Copyright © 2023 Lei Sun and Xiaolong Xu. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

As a key resource for diagnosing and identifying problems, network syslog contains vast quantities of information. And it is the main source of data for anomaly detection of systems. Syslog presents the characteristics of large scale, diverse types and sources, data noise, and quick evolvement, which makes the detection methods not generic enough. To effectively address problem of log anomaly labelling caused by massive heterogeneous logs, we propose LogPal, a generic anomaly detection scheme of heterogeneous logs for network systems, which innovatively combines template sequences and raw log sequences to construct and generate log pattern events. By improving the self-attention mechanism of transformer, LogPal proactively synthesizes self-attention and handles log pattern events in a unique way. The model can make full use of log template and sequence semantic information, by automatically becoming aware of the pattern of logs. We implemented experiments to evaluate the performance of LogPal on publicly available datasets, and the outcome of the experiments shows that LogPal automatically adapts to log type changes and improves precision, recall, and $F1$ score to 99% on publicly available datasets.

1. Introduction

When the system is running, syslog is used to record the runtime state and events of the system, including the anomalies of the system. As the most reliable source of information for monitoring the health of a system, syslog contains massive amounts of information and is the main source of data for anomaly detection in the system [1]. For traditional standalone systems, developers write specific rules based on domain knowledge or manually check logs to detect system anomalies.

However, modern information systems usually adopt a distributed architecture. Syslog is multisourced and heterogeneous. Syslog usually originates from multiple subsystems with various types, structures, implementations, versions, and deployment environments [2, 3]. The approach to anomaly detection, which relies heavily on manual check of logs, is almost unworkable for large-scale system. Moreover, developers usually use free text to record system time for convenience and flexibility. Examples of heterogeneous logs are shown in Table 1.

More importantly, just like any other software maintenance, syslog is constantly evolving. Developers may frequently modify the source code, including logging statements. So, this can create a new log pattern that has not appeared and affected the results of anomaly detection. As Kabinna et al. [4] observed, in their research project, about 20%~45% of the logging statements changed during their lifecycle. Many new log events and log sequences are generated by dynamic logging statements.

Therefore, many automated anomaly detection methods based on logs have been proposed in recent years, and these methods are mainly classified into unsupervised learning and supervised learning. Unsupervised learning methods usually use machine learning techniques such as clustering and PCA [5–8], but unsupervised learning tends to be less

Table 1: Examples of heterogeneous logs.

Log type	Detailed message
Hadoop	2015-10-17 15 : 38 : 05,258 INFO [main] org.Apache.hadoop.metrics2.impl.MetricsSystemImpl: MapTask metrics system started
Thunderbird	2005.11.09 #8# Nov 9 12 : 20 : 55 #8#/#8# sshd[16228]: password authentication for user #41# accepted
Blue gene (L)	2005-06-03-15.42.50.363779 R02-M1-N0-C : J12-U11 RAS KERNEL INFO instruction cache parity error corrected

accurate compared to supervised learning methods. Supervised learning methods generally learn the anomaly patterns of logs based on anomaly labelling to achieve the purpose of anomaly detection. And supervised learning methods usually use deep learning methods such as LSTM and CNN [9–12]. Although some of the above methods can effectively detect anomalies, log sequence anomaly detection problems face the following challenges:

(1) It is rather difficult to achieve a balance between learning log templates and raw log semantic information. Thanks to the rapid development of natural language processing and deep learning, some methods build log anomaly detection methods based on raw log sequences when solving the heterogeneous log anomaly labelling problems, and there is hardly any parsing of the raw log sequences, making it difficult for models to utterly learn log word vector semantics or patterns. There are also approaches that parse the raw logs by extracting log sequence templates and use the log templates as input to build template-based anomaly detection network models. However, these approaches simply using log template sequences as training data to obtain word vectors for the templates, ignoring the key textual information specific to the raw logs, which can lead to more serious results. For example, two or more normal log sequences and anomalous log sequences are considered the same template by removing the critical variable part, and the model "considers" log sequences with different labels as the same input, which is quite fatal for anomaly detection. Therefore, how to make the model understand the log patterns more easily while retaining all the information of log semantics becomes one of the key issues for log-based anomaly detection.

(2) There is a large amount of noise in log data. A certain level of noise is inevitably interspersed in the collection and preprocessing of log data [13]. Log data are derived from various events that occur on distributed hardware and software systems. These events include both events that characterize the system as anomalous, such as being subject to DDoS attacks, storage failures, anomalous system behavior, and network jitter, and events that characterize the system as normal, such as successful ping sessions, successful subsystem startups, and file reads and writes. Since logs are usually generated by multiple processes or threads of the system, a log sequence often contains multiple normal/anomalous. This results in an anomalous log sequence often interspersed with one or more normal logs, presenting a significant challenge for log sequence anomaly detection. In addition, in large-scale systems, many logs are generated individually by geographically distributed components and then uploaded to a centralized location for further analysis. This collection process can lead to missing, duplicated, or disordered log sequences (e.g., due to network errors, limited system throughput, storage issues, etc.) [14]. A McKinsey network survey [15] found that 80% to 98% of logs are just noise, which makes processing and analyzing log data tricky. Noise in log data hinders the effectiveness of existing log-based anomaly detection methods.

(3) Accuracy and recall are still difficult to balance. In anomaly detection based on heterogeneous logs, the precision rate refers to the proportion of true anomalous logs among those predicted to be anomalous; the recall rate refers to the proportion of logs that are predicted to be anomalous among all true anomalous logs. As we all know, there exists a relation of "as one falls, another rises." It is an uphill battle to have both accuracy and recall. The system can generate hundreds of millions of system logs in just a few months, among which the anomalous logs can reach hundreds of thousands; even if there is a 1% error in the precision rate, there may be thousands of false positives, which is a great vexation for operations staff. Likewise, if the recall rate has 1% error, this means that there will be thousands of anomalous logs ignored, and some of them may be caused by fatal failures, which will cause serious losses. How to balance and improve the two is one of the most important challenges for researchers to overcome today.

To solve the above key challenges, in this study, we propose a generic anomaly detection mechanism for heterogeneous logs, called LogPal, which filters the raw system logs, then uses the FT-tree method to parse the log templates, and next splices the templates with the raw logs to generate log pattern events, thus realizing the automatic parsing of heterogeneous logs. Moreover, based on the semantic similarity of the anomalous sequences of heterogeneous logs, we combine natural language processing methods and deep learning methods to improve the transformer model to learn log patterns more adaptively and effectively to achieve anomaly detection of heterogeneous logs. The contributions of this study can be summarized in the following points:

(1) To address the difficult problem of balancing log templates and all semantic information of the raw

logs, a new log pattern event generation method for heterogeneous logs is proposed, which first filters the log sequences for noise reduction, then uses the FT-tree for template extraction, and then innovatively combines the filtered log sequences to build log pattern events, and the combined pattern events will consist of two parts (template number and filtered real logs).

(2) For the two parts that are different from each other by log pattern events, after embedding the pattern events into log pattern vectors, the synthetic attention approach is prospectively used to improve the transformer model to process log pattern events differently, so as to build a pattern-aware learning model for heterogeneous logs.

(3) To address the large amount of noise present in log sequences, in the synthetic attention part, the model's capability and computational complexity are balanced by the relative deviations of different tokens. The input tokens focus on each token, thinning out Tokens with different deviations away from it in a fine-to-coarse fashion, as a way to reduce or even ignore noise in the log sequence.

The rest of the study is organized as follows. Section 2 analyzes the work related to log-based anomaly detection. In Section 3, we introduce the framework of LogPal and the workflow of log parsing and anomaly detection in detail. Section 4 describes the experimental environment and datasets, evaluation indicators, experimental results, and the corresponding analysis. Section 5 concludes the study and looks forward to future work.

2. Related Work

The traditional machine learning approaches are playing an increasingly influential role in log anomaly detection. For example, Bodik et al. [16] use regression-based analysis techniques to automatically classify and identify performance crises by constructing a new representation of data center state, called a fingerprint, which is constructed by statistical selection and summarization of hundreds of performance metrics typically collected on such systems. It can be used to detect specific performance crises that have been seen before, but has limited effects on new unseen performance crises.

Chen et al. [17] proposed a decision tree learning method to diagnose failures in large Internet sites, which is the first application of decision trees to anomaly detection. The method records the runtime attributes of each request and applies automated machine learning and data mining techniques to determine the cause of failure. The algorithm was able to successfully identify 13 of the 14 true causes of failure, achieving a 93% identification rate.

Although effective, traditional machine learning methods often require manual extraction of features from the raw logs, and the results of the model output depend heavily on the extraction of features. In addition, traditional machine learning methods cannot effectively address the heterogeneity and evolution of logs, making the accuracy of anomaly detection based on traditional machine learning methods not very high. With the rapid development of deep learning and natural language processing, research has focused on the application of sequence-based [9–12, 18–21] models. Du et al. [9] designed the DeepLog framework using LSTM neural networks to realize online anomaly detection on system logs. DeepLog uses not only log keys, but also metric values in log entries to detect anomalies, and it relies only on a small training dataset consisting of "normal log entries." The LogMerge anomaly detection method proposed by Zhang et al. [13] combines LSTM and CNN methods to effectively extract the backward and forward dependencies of log sequences, yet significantly reduces the impact brought by noise in log sequences. LogMerge learns the semantic similarity of multisyntax logs, which enables the migration of log anomaly patterns across log types and greatly reduces the anomaly annotation overhead. LSTM with attention mechanism has also been used to improve the performance of complex sequence modeling tasks, such as those for which Zhang et al. [14] proposed the anomaly detection method LogRobust. LogRobust extracts semantic information of log events and represents them as semantic vectors. Then, it detects anomaly using an attention-based bi-LSTM model that captures contextual information in log sequences and automatically learns the importance of different log events. In this way, LogRobust can identify and handle unstable log events and sequences, is robust to unstable log data, and solves the problems of unstable log data in anomaly detection, but when the log sequences span is large and the network is deep, it can greatly increase the calculation. These are some explorations of log sequence anomaly detection with LSTM, but further improvements are needed in detecting accuracy and reducing computational overhead.

Transformer [22] is a state-of-the-art NLP architecture based on self-attention, it breaks the limitation that LSTM models cannot be computed in parallel, and the self-attention mechanism is a more interpretable model that has achieved many impressive results on natural language processing tasks, and in recent years, gradually more and more researchers have been applying this model to the field of log anomaly detection. For example, Nedelkoski et al. [18] proposed Logsy, a classification-based method to learn log representations that allow to distinguish between normal system log data and anomaly samples from auxiliary log datasets, easily accessible via the Internet. The idea behind Logsy is that the auxiliary dataset is sufficiently informative to enhance the representation of the normal data, yet diverse enough to regularize against overfitting and improve generalization. Steverson et al. [19] detect attacks on an enterprise network by applying mining NLP techniques to Windows Event Logs (WELs), using transformer models and self-supervised training methods. A self-supervised anomaly detection model was constructed by combining deep learning methods, traditional machine learning, and natural language processing. The model filters log into a series of words with a few simple steps. The model does not perceive template for input and has poor generalization

ability to logs of the same template that have not appeared, in addition to the simple filtering of logs makes it difficult to eliminate the effect of log noise and may even make log data noisier. Le and Zhang [23] proposed NeuralLog, a novel log-based anomaly detection approach that does not require log parsing. NeuralLog extracts the semantics from raw log sequences and represents them as semantic vectors. These representation vectors are then used to detect anomalies using a transformer-based classification model.

There are other deep learning methods for log anomaly detection. Qi et al. [24] proposed a novel log-based anomaly detection method called Adanomaly, which uses the BiGAN model for feature extraction and an ensemble approach for anomaly detection. Han et al. [25] proposed a data augmentation strategy that generates a set of anomalous sequences by negative sampling so that practitioners can use the observed normal sequences and the generated anomalous sequences to train a binary classification model.

3. Classification-Based Log Anomaly Detection

3.1. Framework. To address the challenges brought by the heterogeneity, evolution, and data noise of logs, we propose LogPal for generic anomaly detection for heterogeneous logs under massive noise. LogPal can automatically parse heterogeneous logs and improve the accuracy of syslog anomaly detection by combining the raw logs to obtain the final log pattern events, and LogPal can sense the log patterns through an improved transformer model to achieve anomaly detection. This section describes the overall framework of LogPal and the details of each part.

Figure 1 shows the overall framework of LogPal, which is divided into two modules: the offline training module and the online detection module. In the offline training module, LogPal first uses the FT-tree method to extract templates from the raw logs, and the templates are combined with the raw logs to parse them into new log pattern events, and construct pattern vectors based on the log pattern events. LogPal inputs the pattern vectors into the transformer deep neural network model of synthetic attention and trains a general anomaly detection model for heterogeneous logs. In the online detection module, LogPal maps online log sequences to pattern vectors based on the above method, judges whether an online log sequence is anomalous according to the trained anomaly detection model, and generates an alarm if it is an anomalous log sequence.

3.2. Pattern Vector Construction. Syslog is usually an unstructured natural language text written by different developers and often needs to be parsed by log parsers before it can be effectively applied for anomaly detection based on machine learning, deep learning, and other methods. Currently, it is a common practice to parse syslog by extracting templates from the syslog. A template is usually an invariant part of the syslog that represents the general type and meaning of the event expressed by the log sequence, and similar log sequences can be represented by the same templates, e.g., "** startup succeeded" is "syslog: klogd startup succeeded" which is a template for "syslog: klogd startup succeeded." Compared with the raw log, the template removes the variable part "syslog: klogd" and keeps the main part of the event, i.e., "A process or port started successfully." This template can represent not only the log sequence "syslog: klogd startup succeeded," but also other log sequences that describe the same event as this log sequence, such as "syslog: syslogd startup succeeded."

We use the FT-tree template parser [26] for template extraction. FT-tree is an extended prefix tree structure with the basic idea that a fixed part of a log sequence is usually the longest combination of frequently occurring words. Therefore, extracting templates is equivalent to identifying the longest combination of frequently occurring words from the logs. Numerous experiments based on production environment logs show that FT-tree supports incremental learning with high accuracy and high template matching efficiency. However, simply taking log template sequences as training data and constructing template vectors based on them, although effective, ignores key textual information peculiar to the raw logs, which results in two or more normal and exception log sequences, removing the critical variable parts, and generating the same template. This makes the model "think" of log sequences with different labels as the same input, which is fatal for the log anomaly detection model.

In the end, we adopt the frequently used textual preprocessing library torchtext, which filters abundant numbers and special character noise in the raw log sequences and applies character case conversion, then uses FT-tree for template extraction. The extracted log template sequences are encoded as natural number sequences from 1 to n, and each number represents the type of each template. So far, the raw log sequences have been transformed into template tag sequence, and finally new textual token sequences are generated and combined with the raw syslog. A combined pattern event will be composed of two parts (template number and filtered syslog). The new textual tokens sequences not only abstract the main part of each log sequence but also fully retains all the key information of the variable part. In addition, to preserve the semantics of the two parts of log pattern events and reduce or even eliminate the impact of heterogeneous log anomaly detection, LogPal uses all log pattern event tokens (template numbers arranged before syslog sequences) as training data to obtain word vectors of template words and raw syslog sequences and constructs pattern vectors based on them. GloVe [27] integrates latent semantic analysis based on singular value decomposition and the word2vec algorithm by introducing co-occurrence probabilities matrix, which uses both global statistical features of the corpus and local context features. GloVe uses the lexical co-occurrence statistics to change their weights in the objective function J, which is specified as follows:

$$J = \sum_{i,j}^{N} f\left(X_{i,j}\right)\left(v_i^T v_j + b_i + b_j - log\left(X_{i,j}\right)\right)^2, \quad (1)$$

where v_i and v_j are the word vectors of words i and j, b_i and b_j are two deviation terms, f is the weight function, and N is

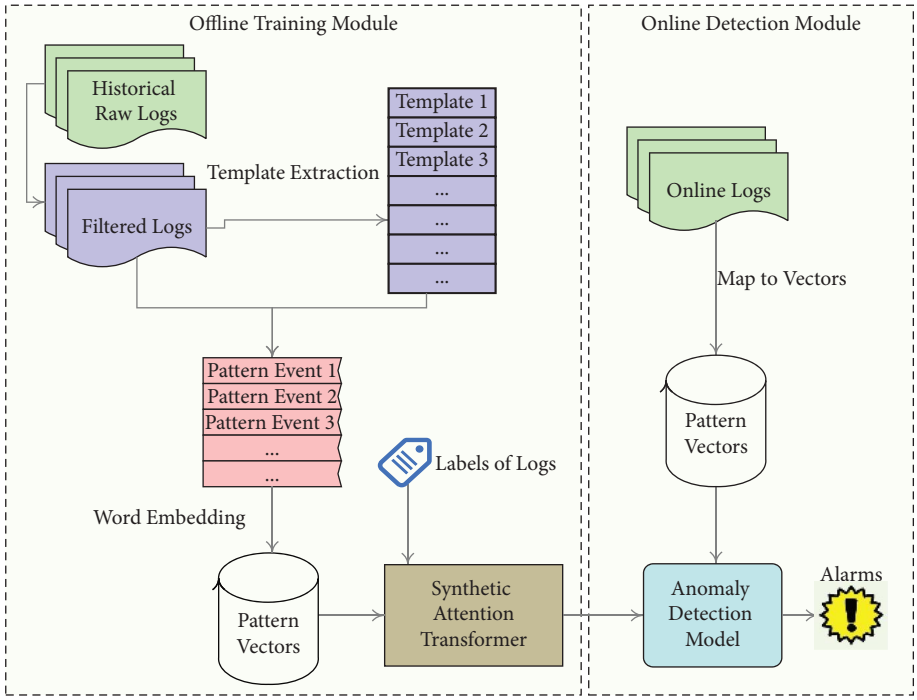


Figure 1: The framework of LogPal.

the size of the vocabulary table (co-occurrence matrix dimension is $N \times N$). The pattern vectors of log pattern events can be obtained by using GloVe. To facilitate the reader's understanding, Figure 2 shows the process of transforming the raw logs into new pattern vectors.

3.3. Synthetic Attention Transformer. LogPal is modeled by an encoder with a multihead attention transformer and takes the constructed log pattern vectors as input, which differs from the input of a traditional transformer in that each pattern vector contains two parts of tokens (the template number and the filtered real log). Therefore, an improved transformer for synthesizing attention is designed to learn the constructed log pattern vectors more efficiently.

Synthetic attention is represented by a synthetic attention matrix, which is divided into global attention and sparse attention. Global attention is applied to the log template, and sparse attention is applied to the log sequence. The log template pays attention to every token of log pattern, including the log template itself, because it can even directly determine the anomaly itself.

However, not every token needs to deal with contextual representation. In the typical self-attention mechanism, every token needs to attend all other tokens; however, for a trained transformer, the learned attention matrix K is usually very sparse at most data points. Therefore, the computational complexity can be reduced by combining structural biases to limit the number of keyword key pairs per query. For a given input token, we can group its contexts into nonoverlapping spans of different sizes, and the size of the spans increases with their relative distance. That is, the input token attends each token, processing the different spans away from it in a fine-to-coarse fashion. To obtain the synthetic attention keyword matrix, the template token attention and the sparse log token attention are constructed successively.

3.3.1. Template Global Attention. Global attention is used for the template token of the constructed log vector, "global" means that the template token can both attend all other tokens and let all other tokens pay attention to it. The attention formula is as

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{2}$$

where $\text{Attention}(Q, K, V)$ is the value of attention and Q, K, and V are the query vector matrix, key vector matrix, and value vector matrix, respectively. Every row of these three matrices represents a vector corresponding to a token, and we need to calculate a Score Matrix for the template vector before calculates the template attention:

$$\text{Score} = \text{QK}^{\text{T}}. \tag{3}$$

Templates are very important for anomaly detection, so global attention is applied to templates. That is, only the attention between the template token and other tokens, and the attention of other tokens with the template token are calculated. Figure 3 illustrates this process.

3.3.2. Log Sparse Attention. Unlike templates, each token of log sequences is processed from center to both ends. Every token pays more attention to the log sequence token that is closer to itself, and the further distant token is not as

Raw Log Sequences:
Log1: syslog: klogd startup succeeded.
Log2: syslog: syslogd startup succeeded.
Log3: MapTask metrics system stopped.
Log4: MapTask metrics system shutdown complete.

Templates → S/N:
Template1 →1 : * * startup succeeded
Template2 →2 : maptask metrics system * *

Pattern Tokens:
Pattern1: 1 syslog klogd startup succeeded
[35, 1140, 805, 832, 3577, 1]
Pattern2: 1 syslog: syslogd startup succeeded
[35, 1140, 706, 832, 3577, 1]
Pattern3: 2 maptask metrics system stopped
[543, 1856, 653, 4551, 56, 1]
Pattern4: 2 maptask metrics system shutdown complete
[543, 1856, 653, 4551, 56, 1860]

Pattern Vectors:
Pattern Vector1: [[-0.0411, -0.0023,…], [-0.0310, 0.0423,…], […]…[…]]
Pattern Vector2: [[-0.0411, -0.0023,…], [-0.0310, 0.0423,…], […]…[…]]
Pattern Vector3: [[0.1334, -0.6031,…], [-0.0654, -0.0630,…], […]…[…]]
Pattern Vector4: [[0.1334, -0.6031,…], [-0.0654, -0.0630,…], […]…[…]]

Figure 2: Examples of mapping raw log to pattern vector.

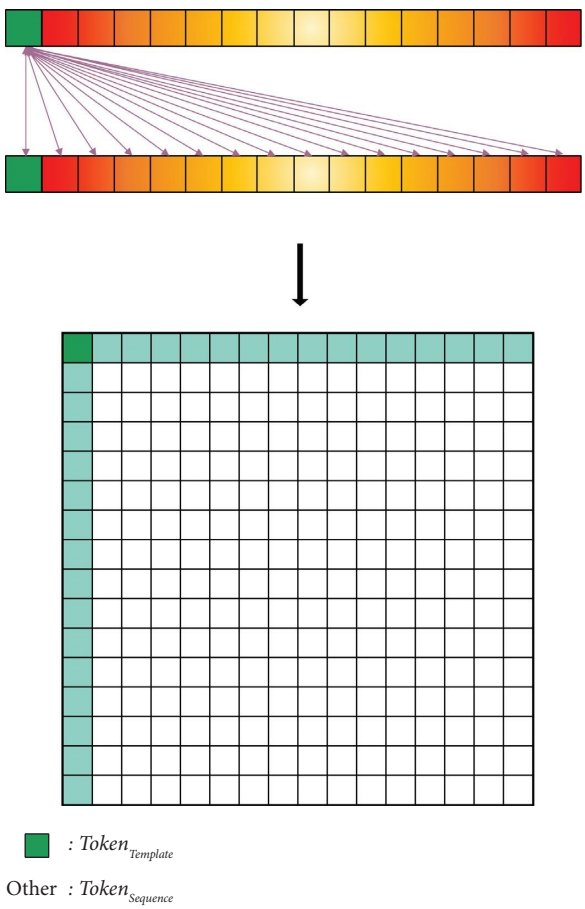


Figure 3: Global attention process of template token vector.

concerned, which can significantly decrease the subsequent parameters in quantity by sparsity. The following Q matrix, K matrix, and Score matrix all only describe the raw log sequence tokens, without the template tokens.

In the K matrix, we take a heterogeneous log sequence with m tokens as an example. For a certain token_i, the distance deviations of every raw log token_j (without considering the template token) from token_i is calculated as follows:

$$\text{Deviation}_j = |i - j|, \tag{4}$$

and then, we input m deviations to the minimum heap

$$\text{MinHeap}_{\text{Deviation}} = \{\text{Deviation}_0, \text{Deviation}_1, \ldots, \text{Deviation}_{m-1}\}. \tag{5}$$

By inputting the deviation into the minimum heap, we can ensure that the next selected token has the minimum deviation from token_i. And then, LogPal selects several groups of tokens from the minimum heap, and the number of tokens in each group is 2^0, 2^1, 2^2, ..., total N tokens. The vector in each cluster takes the maximum value. Next, the maximum vector is used to calculate the Score vector. Then, process each token_i in sequence to get the sparse matrix W^{Score}. The ith row and jth column of the raw log vectors (excluding the template vector) are the sparse attention values of token_i with its own and other token, and finally further calculated by equation (2) to obtain the sparse attention matrix from fine to coarse. This is based on the assumption that for any token, the nearest token requires more attention, while the distant token has little impact on it. This can reduce the effect of noise away from the distant token. At the same time, it also decreases the parameter quantity of subsequent calculations. Finally, the sparse attention matrix obtained in part 2 is spliced to the lower right of the template global attention matrix.

Figure 4 illustrates the mapping process from the Q matrix and K matrix to the *Score* matrix with the raw log sequence whose length of token is 15 as an instance. token_7 and token_8 are shown in Figure 4.

The pseudocode of the sparsity algorithm is shown in Algorithm 1.

3.4. Parameter Setting. In the online detection module, every pattern tokens in the log pattern events is mapped to a 300-dimensional vector in the same way, 4 heads are used for multihead attention, a cross-entropy function is used as the loss function to train the LogPal neural network, and a Dropout layer is used to prevent overfitting, a sigmoid layer is employed to output the classification, and we use a weight decay factor 0.001, the initial learning rate is set to 0.001 for the Adam optimizer, and the final training epoch is set to 10. In addition, the random seed can be initialized to a fixed value to ensure that the experimental results can be reproduced. Our model is implemented using PyTorch and trained on an NVIDIA GeForce RTX 3090 GPU.

4. Experiments

To quantify the performance of LogPal, we conducted various experiments. We compare this method with four exposed baselines on two real-world syslog datasets. We

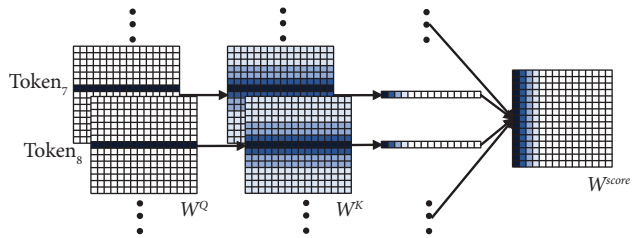


Figure 4: Mapping process from Q matrix and K matrix to sparse score matrix.

Input: W^Q, W^K
Output: W^{score}
(1) Array = $N_1, N_2, N_3, \ldots$ and sum equals Token's numbers;
(2) Min − Heap for offering and polling Deviation;
(3) **for** row_i in W^Q **do**
(4) **for** row_j in W^K **do**
(5) Min − Heap add $|i - j|$;
(6) **end for**;
(7) **for** N_k in Array **do**
(8) **while** not end of N_k **do**
(9) ListKey_k add (Min − Heap poll);
(10) $\text{Key}_k \longleftarrow \text{Max}(\text{ListKey}_k)$;
(11) $W^{\text{score}}_{i,k} \longleftarrow Q_k \cdot K_k$;
(12) **end for**;
(13) **end for**;
(14) returnW^{score};

Algorithm 1: Sparsity of score matrix.

describe the main information in the datasets, discuss the experimental settings and evaluation indicators, and give the results.

4.1. Datasets. We evaluate the proposed method on the following three open log datasets: BGL dataset [28], HDFS dataset [7], and Thunderbird [28]. A brief summary is shown in Table 2, and the details are as follows:

The BGL dataset is an open dataset of logs collected from a BlueGene/L supercomputer system at Lawrence Livermore National Labs (LLNL) in Livermore, California, with 131,072 processors and 32,768 GB memory. The log contains alert and nonalert sequences identified by alert category tags. In the first column of the log, "–" indicates nonalert sequences while others are alert sequences. The label information is amenable to alert detection and prediction research. It has been used in several studies on log parsing, anomaly detection, and failure prediction.

The HDFS dataset is generated in a private cloud environment using benchmark workloads and manually labeled through handcrafted rules to identify the anomaly. The logs are sliced into traces according to block IDs. The HDFS dataset marks each block sequence as normal or anomalous. The HDFS dataset consists of 11,175,629 logs collected in 38.7 hours on more than 200 Amazon EC2 nodes. There are 575,061 log blocks in the dataset, of which 16,838 are marked as "exception" by Hadoop experts.

Table 2: A brief summary of datasets.

Dataset	Time span	Size	No. of messages	Anomalies
BGL	214.7 days	708 MB	4,747,963	348,460
HDFS	38.7 hours	1.47 GB	11,175,629	288,250
Thunderbird	244 days	29.60 GB	211,212,192	43,087,287

Thunderbird dataset is an open dataset of logs collected from a Thunderbird supercomputer system at Sandia National Labs (SNL) in Albuquerque, with 9,024 processors and 27,072 GB memory. The log contains alert and nonalert sequences identified by alert category tags. In the first column of the log, "–" indicates nonalert sequences, while others are alert sequences.

4.2. Experimental Setup. The experimental setup for this study is explained as follows.

4.2.1. Comparisons

We compared LogPal with five published baseline methods, namely, PCA [7], DeepLog [9], Swisslog [29], HitAnomaly [30], and InterpretableSAD [26]. The parameters of these methods have been optimized to

produce their best evaluation scores. A brief description of these methods is as follows:

PCA: in this model, Logs are converted to count vectors and divided into normal and anomaly spaces using a principal component analysis (PCA) algorithm.

DeepLog: a deep neural network using LSTM models the system logs as natural language sequences.

SwissLog: SwissLog combines semantic embedding and time embedding methods to train a bi-LSTM model based on unified attention for anomaly detection

HitAnomaly: a log sequence encoder and a parameter value encoder are designed to obtain their corresponding representations. The hierarchical transformer structure is used to model the log template sequence and parameter values.

Interpretable SAD: the authors propose a data augmentation strategy that generates a set of anomalous sequences with negative sampling so that a binary classification model can be trained based on the observed normal sequences and the generated anomalous sequences.

4.2.2. Evaluation Indicators. To measure the effectiveness of LogPal in anomaly detection, we use precision, recall, and $F1$ score as indicators.

Precision: it is the percentage of true anomalies among all anomalies detected by the approach:

$$PR = \frac{TP}{TP + FP}. \tag{6}$$

Recall: it is the percentage of anomalies among the dataset being detected:

$$RC = \frac{TP}{TP + FN}. \tag{7}$$

$F1$ score: it is the harmonic mean of precision and recall:

$$F1 = \frac{2 \times PR \times RC}{PR + RC}. \tag{8}$$

TP is the number of anomalous log sequences correctly detected by the model, FP is the number of normal log sequences incorrectly identified as anomalies by the approach, FN is the number of anomalous log sequences that are not detected by the approach, and F1 score is used as a metric that considers both precision and recall, which does not favor one metric over another and does not lose scientific validity due to the imbalance problem of the dataset.

4.3. Experimental Results. Firstly, we compare LogPal with transformer on BGL dataset and HDFS dataset. We convert the log sequences of the datasets into log pattern vectors in the same way, and then, these pattern vectors are input into the transformer model, and relevant parameter settings are consistent with LogPal. We conducted the comparison experiment by controlling the ratios of the training set and test set. Figures 5 and 6 show the comparative results of the experiment.

The horizontal axis of Figures 5 and 6 represents the ratios of the training set and test set, and the vertical axis represents the $F1$ score of different anomaly detection models. It can be seen that when the training set ratio of LogPal is large, the optimal $F1$ score of the LogPal method for anomaly detection is 99% and that of transformer model is 98%, which has a weak advantage over transformer; when the ratio of training sets is small, it can better reflect the performance advantages of LogPal. It shows that even if a small amount of training data is obtained from the target syslog, LogPal can extract the key information leading to the normal or anomalous log sequences and can produce accurate prediction even in invisible samples. When the ratio of training sets is 2 : 1, LogPal's anomaly detection rate is 90%, while transformer is 86%, LogPal increased $F1$ score by 4.7%. Even with a large training set, the $F1$ score improves by more than 1%. LogPal uses synthetic attention to perceive the relationship between the template vector and the raw vector differently and obtains a better $F1$ score than the transformer model. The transformer model does not consider this special feature of the raw log template but only considers the self-attention relationship matrix of the raw log sequence itself making a lot of important information ignored, which may lead to false alarm. LogPal can quickly perceive and learn the semantic information of the pattern vectors to improve the accuracy of anomaly detection. In addition, LogPal adopts a sparse attention method for the raw log sequence token, which can reduce the noise impact even in the long text log sequence and adopting a general and unified pattern event extraction method to embed the pattern vectors, as we expect, enable robust representation and accurate anomaly detection even for heterogeneous logs or invisible new logs.

To further evaluate the performance of LogPal, we also evaluated LogPal and baselines on BGL dataset, HDFS dataset, and Thunderbird dataset. We show the overall performance of LogPal compared with baselines in Table 3 and Figures 7–9. Based on the three datasets, generally speaking, LogPal has the best performance, and all evaluation indicators are close to 99%. LogPal can generally filter massive heterogeneous logs to generate pattern vectors and perceive log templates and log sequences, respectively, by synthetic attention. PCA and DeepLog use the index of log template to learn anomalous and normal patterns. It ignores the meaning of log sequences and words, and the actual performance is not high. Although InterpretableSAD performs well on the Thunderbird dataset, where all indicators are balanced and the indicator value is not low, the method does not perform so well on the BGL dataset and the HDFS dataset, which may be related to the fact that the method is not universal and may be more suitable for a certain dataset.

It is worth noting that comparing SwissLog and HitAnomaly, SwissLog has a precision of 97% and a recall of 100% in the experiments on the BGL dataset and the HDFS dataset, while the two indicator values of HitAnomaly are exactly the opposite, it has a precision of 100% and a recall of

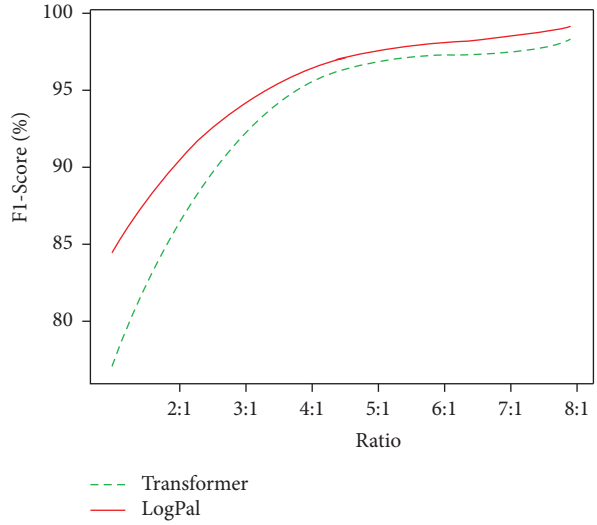


Figure 5: Comparisons of different ratios on the BGL dataset.

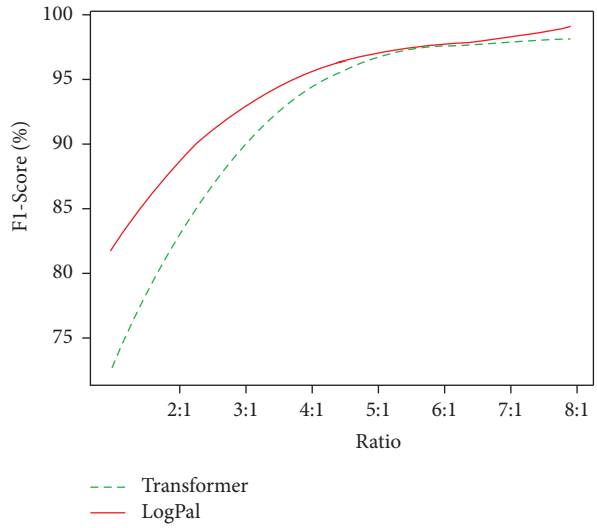


Figure 6: Comparisons of different ratios on the HDFS dataset.

Table 3: Comparison of methods on three datasets.

Methods	Datasets								
	BGL			HDFS			Thunderbird		
	PR (%)	RC (%)	*F*1 score (%)	PR (%)	RC (%)	*F*1 score (%)	PR (%)	RC (%)	*F*1 score (%)
PCA	98	67	80	50	61	55	87	90	89
DeepLog	95	96	93	92	92	91	95	92	93
HitAnomaly	**100**	97	98	**100**	97	98	97	96	97
SwissLog	97	**100**	**99**	97	**100**	98	95	**97**	96
InterpretableSAD	94	88	91	92	87	89	97	96	96
LogPal	99	99	**99**	98	99	**99**	**98**	**97**	**98**

The bold values show the best performances of method based on the experimental results with the specific indicator and dataset.

97% in the experiments. This means that SwissLog is more inclined to detect log sequences as anomalous, in other words, it is more capable of uncovering anomalous logs in the logs. Although the recall rate is satisfactory, it is clear from the precision rate that SwissLog mistakes some log sequences that are really normal as anomalous. Thus, generating a large number of false positive predictions, and if a log anomaly detection method generates too many false alarms, which will consume energies of O&M staff to verify the system condition and add a lot of unnecessary work; in

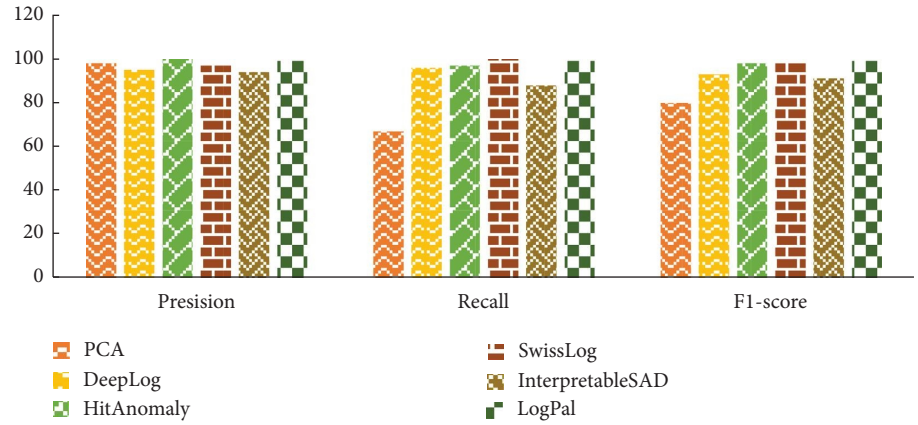


Figure 7: Comparison of different methods on the BGL dataset.

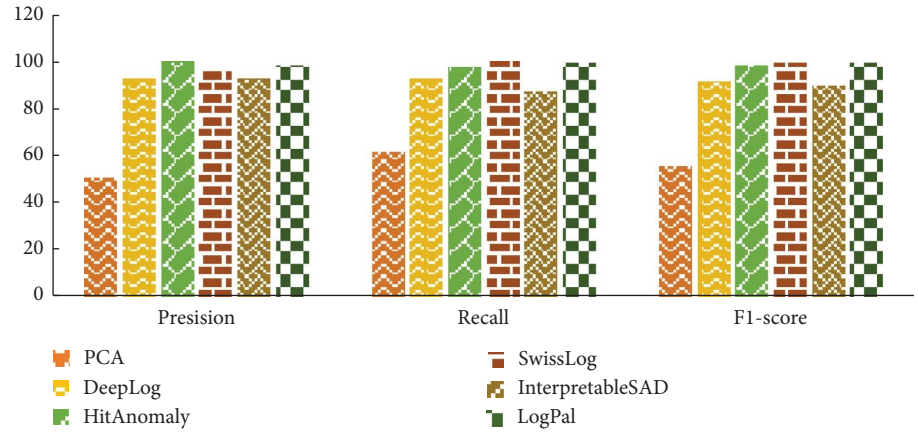


Figure 8: Comparison of different methods on the HDFS dataset.

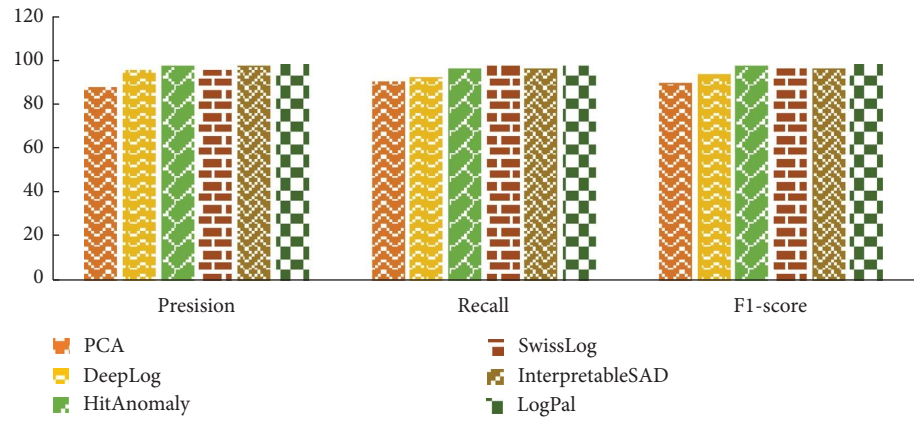


Figure 9: Comparison of different methods on the thunderbird dataset.

contrast, HitAnomaly can be very precise in logs identified as anomalous, without so much false positive predictions. But it misses some anomalous logs, thus generating a large number of false negative predictions, which may be a more serious problem if it fails to detect system anomaly. System failures may not be resolved in a timely manner for a long time, which will cause serious losses. Generally speaking, LogPal is able to balance precision and recall and has an improved overall performance. Overall, compares favorably to baselines, LogPal not only learns the semantic information of log word vectors but also focuses differently on the attention relation between template tokens and log sequence tokens. Finally, LogPal achieves an excellent performance on the BGL dataset, the HDFS dataset, and the Thunderbird

dataset. Compared with existing methods, LogPal improves the $F1$ score by 1% on the HDFS dataset. Besides, LogPal improves the precision and $F1$ score by 1% on the Thunderbird dataset.

5. Conclusion

As a kind of data reflecting system status and events, syslog provides an important support for detecting various software and hardware system anomalies. Many log-based methods have been proposed to detect anomaly in large-scale software and hardware systems. However, the existing methods make it difficult to effectively deal with the labelling problems in heterogeneous logs. To overcome these problems, this study proposes a generic anomaly detection mechanism for heterogeneous logs, called LogPal. The model innovatively utilizes the synthetic attention transformer encoder network, which prospectively thins out the semantics of log sequences and weakens the influence of noise. Compared with other methods, it achieves better generalization ability on multisource and heterogeneous samples. Experiments based on public datasets show that the overall performance of LogPal is better than the current machine learning and deep learning methods. In future work, we will further improve the accuracy of anomaly detection by introducing the weight coefficient to learn the contribution degree of the template and the raw log token. In addition, we will explore the synthesis strategy of synthetic attention to reduce the computational complexity and improve the early warning speed of anomaly detection.

Data Availability

Previously reported log data were used to support this study and are available at https://doi.org/10.48550/arXiv.2008.06448.

Conflicts of Interest

The authors declare that they have no known conflicts of financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported by the National Natural Science Foundation of China, under Grant 62072255.

References

[1] S. L. He, P. J. He, Z. B. Chen, T Yang, Y. Su, and M. R. Lyu, "A A Survey on Automated Log Analysis for Reliability Engineeringurvey on automated log analysis for reliability engineering," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–37, 2021.

[2] S. He, J. Zhu, and P. He, "Experience report: system log analysis for anomaly detection," in *Proceedings of the 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, O. Ottawa, Ed., pp. 207–218, Ottawa, ON, Canada, October 2016.

[3] H. Li and Y. Li, "LogSpy: system log anomaly detection for network systems," in *Proceedings of the 2020 International Conference on Artificial Intelligence and Computer Engineering (ICAICE)*, pp. 347–352, Hangzhou, Zhejiang, China, October 2020.

[4] S. Kabinna, C. P. Bezemer, W. Shang, MD Syer, and AE Hassan, "Examining the stability of logging statements," *Empirical Software Engineering*, vol. 23, no. 1, pp. 290–333, 2018.

[5] J. Chen, J. Y. Ouyang, and A. Q. Feng, "DoS anomaly detection based on isolation forest algorithm under edge computing framework," *Computer Science*, vol. 47, no. 2, pp. 293–299, 2020.

[6] Q. W. Lin, H. Y. Zhang, and J. G. Lou, "Log clustering based problem identification for online service systems," in *Proceedings of the 38th International Conference*, pp. 102–111, Austin, Texas, USA, May 2016.

[7] W. Xu, L. Huang, and A. Fox, "Detecting large-scale system problems by mining console logs," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132, Big Sky, Montana, USA, July 2009.

[8] S. Ying, B. M. Wang, L. Wang et al., "An An Improved KNN-Based Efficient Log Anomaly Detection Method with Automatically Labeled Samplesmproved KNN-based efficient log anomaly detection method with automatically labeled samples," *ACM Transactions on Knowledge Discovery from Data*, vol. 15, no. 3, pp. 1–22, 2021.

[9] M. Du, F. Li, and G. Zheng, "Deeplog: anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference*, pp. 1285–1298, Dallas, Texas, USA, July 2017.

[10] B. Sharma, P. Pokharel, and B. Joshi, "User behavior analytics for anomaly detection using lstm autoencoder - insider threat detection," in *Proceedings of the IOE GC*, T. Bangkok, Ed., pp. 1–9, March 2020.

[11] X. Y. Duan, S. Ying, H. L. Cheng, W Yuan, and X Yin, "OILog: An online incremental log keyword extraction approach based on MDP-LSTM neural network," *Information Systems*, vol. 95, pp. 101618–11, 2021.

[12] S. Garg, K. Kaur, N. Kumar, G Kaddoum, AY Zomaya, and R Ranjan, "A A Hybrid Deep Learning-Based Model for Anomaly Detection in Cloud Datacenter Networksybrid deep learning-based model for anomaly detection in cloud datacenter networks," *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 924–935, 2019.

[13] S. L. Zhang, W. D. Li, and Y. Q. Sun, "Unified anomaly detection for syntactically diverse logs in cloud," *Science CSRD*, vol. 57, no. 4, pp. 778–790, 2020.

[14] X. Zhang, Y. Xu, and Q. Lin, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting*, pp. 807–817, New York, NY, USA, August 2019.

[15] I. Kornoukhov and H. Soller, *Mitigating Cyber Risks with Smart Log Management,* Mckinsey & Company, New York, NY, USA, 2020, https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/tech-forward/mitigating-cyberrisks-with-smart-log-management.

[16] P. Bodik, M. Goldszmidt, and A. Fox, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the European Conference on Computer Systems, Proceedings of the 5th European conference on Computer systems*, pp. 111–124, New York, NY, USA, January 2010.

[17] M. Chen, A. X. Zheng, and J. Lloyd, "Failure diagnosis using decision trees," in *Proceedings of the International Conference*

on Autonomic Computing, 2004. Proceedings, pp. 36–43, New York, NY, USA, May 2004.

[18] S. Nedelkoski, J. Bogatinovski, and A. Acker, "Self-attentive classification-based anomaly detection in unstructured logs," in *Proceedings of the 2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1196–1201, Sorrento, FL, USA, February 2020.

[19] K. Steverson, C. Carlin, and J. Mullin, "Cyber intrusion detection using natural language processing on windows event logs," in *Proceedings of the 2021 International Conference on Military Communication and Information Systems (ICMCIS)*, pp. 1–7, The Hague, Netherlands, May 2021.

[20] R. W. Wibisono and A. I. Kistijantoro, "Log anomaly detection using adaptive universal transformer," in *Proceedings of the 2019 International Conference of Advanced Informatics: Concepts, Theory and Applications (ICAICTA)*, pp. 1–6, Yogyakarta, Indonesia, September 2019.

[21] W. B. Meng, Y. Liu, and Y. C. Zhu, "Loganomaly: unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 4739–4745, Macao, China, August 2020.

[22] A. Vaswani, N. Shazeer, and N. Parmar, "Attention is all you need," in *Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pp. 5998–6008, Long Beach, CA, USA, July 2017.

[23] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proceedings of the 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 492–504, Melbourne, Australia, August 2021.

[24] J. X. Qi, Z. Z. Luan, and S. H. Huang, "Adanomaly: adaptive anomaly detection for system logs with adversarial learning," in *Proceedings of the Network Operations and Management Symposium*, pp. 1–5, Budapest, Hungary, August 2022.

[25] X. Han, H. Cheng, and X. Depeng, "InterpretableSAD: interpretable anomaly detection in sequential log data," in *Proceedings of the 2021 IEEE International Conference on Big Data (Big Data)*, pp. 1183–1192, Orlando, FL, USA, August 2021.

[26] S. L. Zhang, W. B. Meng, and J. H. Bu, "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *Proceedings of the 2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pp. 1–10, Barcelona, Spain, June 2017.

[27] J. Pennington, R. Socher, and C. Manning, "Glove: global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532–1543, Doha, Qatar, January 2014.

[28] A. Oliner and J. Stearley, "What supercomputers say: a study of five system logs," in *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pp. 575–584, Edinburgh, UK, June 2007.

[29] X. Y. Li, P. F. Chen, and L. X. Jing, "Swisslog: robust and unified deep learning based log anomaly detection for diverse faults," in *Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 92–103, Portugal, October 2020.

[30] S. Huang, Y. Liu, C. Fung et al., "HitAnomaly: HitAnomaly: Hierarchical Transformers for Anomaly Detection in System Logierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 8383461, 12 pages
https://doi.org/10.1155/2022/8383461

Research Article

Internet-of-Things-Based Suspicious Activity Recognition Using Multimodalities of Computer Vision for Smart City Security

Amjad Rehman,[1] **Tanzila Saba**,[1] **Muhammad Zeeshan Khan**,[2] **Robertas Damaševičius**,[3] **and Saeed Ali Bahaj**[4]

[1]*Artificial Intelligence & Data Analytics Lab (AIDA) CCIS Prince Sultan University, Riyadh 11586, Saudi Arabia*
[2]*Intelligent Criminology Research Lab National Center of Artificial Intelligence, KICS, University of Engineering & Technology, Lahore, Pakistan*
[3]*Faculty of Applied Mathematics, Silesian University of Technology, Gliwice 44-100, Poland*
[4]*MIS Department College of Business Administration, Prince Sattam Bin Abdulaziz University, Alkharj 11942, Saudi Arabia*

Correspondence should be addressed to Robertas Damaševičius; robertas.damasevicius@polsl.pl

Received 18 April 2022; Revised 17 August 2022; Accepted 29 August 2022; Published 5 October 2022

Academic Editor: Andrea Michienzi

Copyright © 2022 Amjad Rehman et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Automatic human activity recognition is one of the milestones of smart city surveillance projects. Human activity detection and recognition aim to identify the activities based on the observations that are being performed by the subject. Hence, vision-based human activity recognition systems have a wide scope in video surveillance, health care systems, and human-computer interaction. Currently, the world is moving towards a smart and safe city concept. Automatic human activity recognition is the major challenge of smart city surveillance. The proposed research work employed fine-tuned YOLO-v4 for activity detection, whereas for classification purposes, 3D-CNN has been implemented. Besides the classification, the presented research model also leverages human-object interaction with the help of intersection over union (IOU). An Internet of Things (IoT) based architecture is implemented to take efficient and real-time decisions. The dataset of exploit classes has been taken from the UCF-Crime dataset for activity recognition. At the same time, the dataset extracted from MS-COCO for suspicious object detection is involved in human-object interaction. This research is also applied to human activity detection and recognition in the university premises for real-time suspicious activity detection and automatic alerts. The experiments have exhibited that the proposed multimodal approach achieves remarkable activity detection and recognition accuracy.

1. Introduction

In recent years, ever-increasing technological advances have made automated human activity recognition a common research subject. Video surveillance has a wide range of applications. These applications include normal and suspicious activities such as gaming, human-computer interaction, exam invigilation, detecting chaos, analyzing sports, predicting crowd behavior, etc. It is an important safety aspect for indoor and outdoor environments [1].

Innovations are occurring rapidly, and since there is a large amount of video data to process, manual intervention is not feasible and is error-prone. Additionally, it is exceedingly challenging to monitor public spaces constantly. Hence, it is necessary to install intelligent video surveillance that can track people's movements in real time, classify them as routine or exceptional, and provide alerts [2].

Human activity detection relies on sensors like radar, cameras, and cell phones to identify abnormalities in human behaviour. They are being used for human-computer interaction, surveillance, monitoring suspicious activities, and other security purposes [3, 4]. The majority of today's systems rely on video gathered from CCTV cameras. If a crime or act of violence occurs, this footage will be utilized in the investigation. It would be preferable, however, to build a system that might identify an anomalous or

unexpected circumstance beforehand and notify the authorities [5, 6].

In recent years, ever-increasing technological advances have made automated human activity recognition a common research subject. Video surveillance has a wide range of applications. These applications may include normal and suspicious activities such as gaming, human-computer interaction, exam invigilation, detecting chaos, analyzing sports, predicting crowd behavior, etc. It is an important safety aspect for indoor and outdoor environments [7, 8].

Currently, innovations are occurring at a rapid pace. The most popular exploration topic these days is robotized human activity recognition. Since there is a large amount of video data to process, manual intervention will not only be tiring but also cause omissions, making the system effective and error-prone. Automatic video surveillance has tacked on this issue. It is impossible to monitor CCTV events manually. Whether the event has already occurred or not, searching for the desired event through recordings is extremely time-consuming. However, a system that automatically senses any irregular or abnormal condition in advance and alerts the appropriate authorities is more appealing. It can be used in indoor and outdoor settings [9, 10].

Different efficacious algorithms are used for automatic activity recognition on roads, airports, educational institutions, offices, etc. Computer vision has provided machines with humanlike vision. Large datasets are accessible and can be trained with GPUs' to help make future predictions. Computer vision technology has a few stages, like taking input from surveillance cameras, separating the frames, classifying and labeling the activity, and writing its description. Normally, two types of classification techniques are used in computer vision. Supervised and unsupervised; supervised classification requires manual labeling whereas unsupervised is completely computer-based and does not need computer intervention [11, 12].

Deep learning is the most exemplary architecture that learns difficult tasks among other architectures. It extracts features from images automatically and portrays significant information about the image. Since it extracts features automatically, it makes it more convenient to use. CNN learns visual patterns directly from pixels [13, 14]. Long short-term memory (LSTM) models can be used for videos as they can recall things for a longer time. The proposed work implemented the YOLOV4 for detecting the different activities related to surveillance and for recognizing the activities, 3D CNN is used. Multiple cameras are connected to the centralized system via IoT (Internet of Things) protocols. Ethernet communication creates a local server to access each camera feed through its specific IP address used in the centralized GPU for prediction [15, 16].

The remaining paper is organized as such: Section 2 explores the relevant state of art critically and highlights the need for this research. Additionally, the main contributions are also mentioned. Section 3 presents the proposed methodology; Section 4 exhibits results and analysis at length and datasets used for experiments. Finally, Section 5 concludes the research.

2. Background

Understanding human behaviour is now one of the most significant areas of computer vision research. Human activity identification uses data from sensors, such as a sequence of RGB camera images, range sensors, or other sensing modalities [17, 18], to automatically identify and understand human actions. Its applications include surveillance, video processing, robotics, and a variety of systems involving human-computer interaction [19, 20]. In the early 1980s, depth sensors improved human activity recognition. Previous research has concentrated mainly on understanding and identifying behaviors from visible light video streams. Several survey articles summarized these works at various depths and perspectives [21].

Zhu et al. [22] used motion information and contextual features for activity detection in a scene, arguing that actions have a close relationship with context. Following the identification and segmentation of behavior, a two-layered conditional random field is used to recognize events from segmented patterns and contextual knowledge. However, they did not use a benchmark dataset, nor accuracy compared to the reported literature. Yue et al. [23] compared two CNN architectures for integrating color and optical flow data for action recognition using the LSTM network. They claimed higher performance on the Sports 1 million dataset (73.1% vs. 60.9%) and the UCF-101 datasets with (88.6% vs. 88.0%) and without (88.6% vs. 88.0%) and without additional optical flow information (82.6% vs. 72.8%). Ibrahim et al. [24] came up with a two-stages of time model to look at unusual activities in the community. They made an LSTM model to show how a person acts in a series of frames, while a second LSTM network adds up the representations at the individual level. Finally, they reported an 81.5% detection accuracy. Khaire et al [25] used the CNN classifier with skeletal data to recognize the different human activities. They used two datasets and achieved 95.11% and 96.67% accuracies. Mariem et al. [26] designed a model named the history of binary motion image (HBMI). In this model, they introduced a new method for foreground detection using the Gaussian mixture model (GMM), including the Magnitude of Optical Flow (MOF). To avoid irrelevant motion, they utilized the fast frame skipping method. Hence, HBMI is a novel method of portraying instructive notions for human activity recognition based on the superposition of human shape. HBMI achieved 97.60% accuracy in the testing state. Xing et al. [27] designed a system to detect driver activities using a deep learning approach. With the help of a low-cost camera, the actions of ten drivers have been recorded. The extracted images are then segmented with a Gaussian mixture model (GMM). These preprocessed images are applied to train AlexNet [28], GoogLeNet [29], and ResNet-50 [30] on activities like texting, mirror checking, using mobile phones, etc. Among these models, AlexNet outperformed the other models with 81.6%, while GoogleNet and ResNet scored 78.6% and 74.9%, respectively. By working on static images only, Chang et al. [31] enhanced the approach of integrated 3D data of human body movements to create a three-dimensional motion history image. Xiaofei et al. [32] suggested a spatiotemporal silhouette representation to describe motion properties, including regular activities. Finally, multiclass SVM

was utilized, with each operation consisting of many views and scenarios of motion descriptors. On the KTH dataset, they attained an average accuracy of 94.10%. In order to simulate the temporal distribution of players in a sporting event and foretell the future course of action, Zhong et al. [33] used hierarchical LSTMs for the temporal encoding of extracted features from video frames and trajectory data. However, no accuracy was reported.

Recently, Saba et al. [34] detected anomalies in smart hospitals by using principal component analysis (PCA) for activity feature extraction. Finally, an ensemble classifier is employed for anomaly classification. Experiments were performed on the KDDCup-"99" dataset and 93.2% accuracy was reported. Patalas-maliszewska et al. [35] adopted CNN, Support Vector Machine (SVM), and CNN region-based CNN (Yolov3 Tiny) for recognizing completed work tasks in the industrial environment. The work of González et al. [36] was heavily focused on achieving real-time results. They conducted extensive research utilizing various datasets and trained Faster-RCNN using Feature Pyramid Network with Resnet50, and outperformed by 3.91 percent as compared to reported techniques in the literature. Bhatti et al. [17] extracted data from YouTube CCTV videos/GitHub repositories and used two approaches (sliding window/classification and region proposal/object detection). They tested several pretrained deep learning classifiers; however, Yolov4 had the best performance for detecting suspicious activity, with an F1 score of 91% and an average accuracy of 91.73%.

Based on the literature analyzed, it is concluded most of the research conducted did not consider a number of expected instances, mostly used static images, and was tailored for specific purposes. However, the final aim of the proposed research is to use the automatically identified behaviors and activities in groups in live videos. Hence, in the proposed research work, we first detect the area of interest and then pass it to the classification network. The reduction of unnecessary learning information increased efficiency and accuracy.

This study has the following main contributions:

(1) A novel activity recognition and detection framework utilizing the YOLOv4 version and the 3D-CNN.

(2) Fine-tune convolution neural network architectures for better object recognition accuracy by incorporating object spatial and temporal information.

(3) Internet of Things-based architecture has been utilized to incorporate and manage the decision-making of deep learning-based architectures efficiently.

3. Proposed Methodology

The proposed methodology is based on two steps. First, we detected the region of interest (ROI) using the fine-tuned version of the Yolo-v4. Secondly, a sequence of 16 frames is generated and ROI is passed through a sequence of frames into the 3D-CNN for classification.

3.1. Activity Detection. The YOLOv4-tiny [37] object detector is a light version of the YOLOv4 [38], enhancing the detection speed. With this light version, YOLOv4 can attain around 370 frames per second (FPS) with very good accuracy on a GPU-enabled machine having a 1080Ti GPU. The YOLOv4-tiny includes cross-stage partial connections (CSP) Darket53-tiny as the backbone feature extractor instead of CSPDarket53, which was used in the original YOLOv4 [38]. The YOLOv4-tiny network uses a cross-stage partial block as a residual block, enhancing the accuracy but increasing the model complexity and eventually decreasing the FPS rate. A tradeoff is to proceed with object detection in real time on embedded devices with better accuracy. Therefore, an improved version of YOLOv4-tiny is proposed.

Figure 1 exhibits the enhanced Residual block (ResBlock instead of two CSPBlock as in YOLOv4-tiny to improve processing speed. The Enhanced ResBlock unit uses two direct path networks to handle the input representation map. In this two-path network, the path T network has three 1×1 and 3×3 convolutional (Conv) layers with stride 2, followed by another 1×1 Conv layer. Another network, Path B, has two 3×3 max pooling with stride 3 followed by a 1 x Conv layer. Compared to CSPBlock used on the original YOLOv4-tiny [37], the proposed ResBlock removes the first 3×3 Conv in CSPBlock and replaces the consequent 3×3 Conv layers with 1×1 Conv layers in the Path T network to make the detection network efficient as exhibited in Figure 1. The proposed ResBlock unit adds pooling and Conv in the Path B network. Still, this extra computation overhead is minimal as compared to reduce in computation in the Path T network. The floating-point operations (FLOPs) are analyzed to determine the computational complexity of the CSPBlock [37] and the proposed ResBlock. FLOPs can be described as follows:

$$FLOPs = \sum_{l=1}^{S} M_i^2 . F_i^2 . C_{i-1} . C_i. \tag{1}$$

Here, S is the sum of all the Conv layers, M_i^2 is the output feature vector of the corresponding lth layer, F_i^2 is the filter size, while C_1 and C_{i-1} refer to output and input channel count, respectively. For comparison, suppose an input of 224×244 with 64 channels, and using (1), FLOPs of ResBlock are used in the proposed detection model as shown in calculations in equations (2) and (3)

$$\begin{aligned} FLOPs &= 104^2 * 1^2 * 64 * 32 + 52^2 * 3^2 * 32^2 \\ &\quad + 52^2 * 1^2 * 32 * 64 + 64 * 52^2 * 2^2 + 52^2 * 1^2 * 64^2, \end{aligned} \tag{2}$$

$$FLOPs = 6.4x10^7.$$

The FLOPs of CSPBlock are used in YOLOv4-tiny against the same image

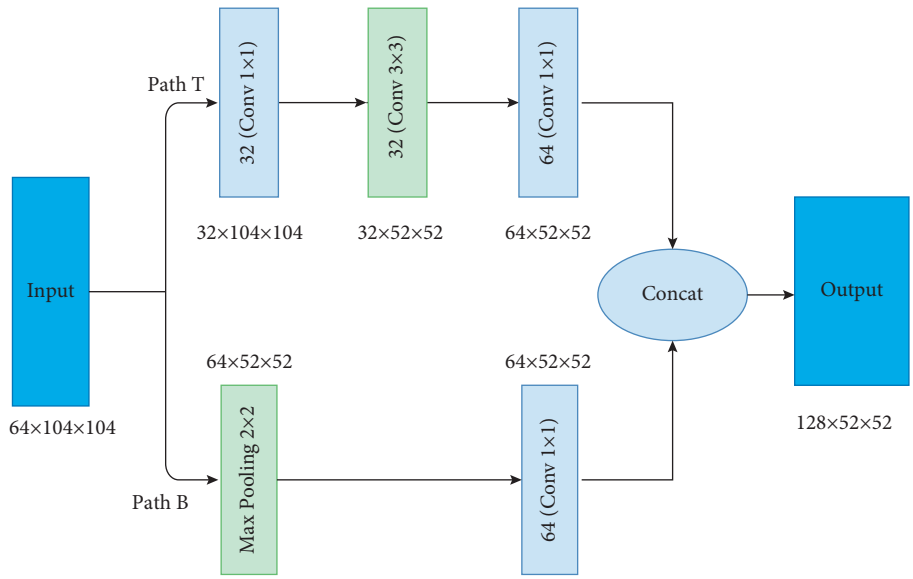


Figure 1: Enhanced ResBlock-D modules.

$$FLOPs = 104^2 * 3^2 * 64^2 + 104^2 * 3^2 * 64 * 32 + 104^2 * 3^2 * 32^2 + 104^2 * 1^2 * 64^2, \quad (3)$$
$$FLOPs = 7.4x10^8.$$

From equations (2) and (3), we determine that 1 : 10 is the computation of FLOPs in ResBlock and CSPBlock. FLOPs comparison shows that ResBlock is much less complex than CSPBlock.

Although the inclusion of ResBlock in the YOLOv4-tiny detector makes it much faster than CSPBlock, it affects object detection accuracy. Therefore, two auxiliary residual blocks are also built and included in the ResBlock unit to get a better tradeoff between efficiency and accuracy. The proposed backbone network is shown in Figure 2.

The output representation of ResBlock is fused with a shallow representation of the backbone model through an element-wise sum operation. This fused representation is used as input to successive layers of the backbone model. The fusing process of representation of ResBlock and the backbone model can be expressed as

$$O^i = f^i\left(O^{i-1}\right) + Or^i. \quad (4)$$

Here in equation (4), i is the index of the layers, f^i is the fusion function between the input and output in the ith layer network, O^{i-1} refers to the i-1th layer's output and the i th layer's input, and Or^i is the output of the proposed ResBlock. This fusion catalyzes the convergence between deep and shallow networks. Moreover, with the fusion mechanism, the network learns more information to enhance the accuracy while preventing the large step-sized calculation increase.

In the backbone of YOLOv4-tiny [37], the Residual network module uses 3×3 filters for feature extraction. Although 3×3 receptive fields can extract more localized information while losing global contextual information and eventually reduces the detection accuracy. We have compensated for this loss of global representations by using two consecutive 3 x Conv layers to get the receptive field of size 5×5 in the auxiliary ResBlock. This auxiliary model passes on the obtained global representation to the backbone network. Then the backbone network joins the local contextual information extracted from the smaller (3×3) receptive field and global representation extracted from the bigger (5×5) receptive field that gives extra information about the object. This combining of global and local information not only enhances the network depth but also advances the semantic of information. The attention mechanism can process and transmit the crucial feature and eliminate the invalid features through channel suppression. We have introduced spatial and channel attention modules in the auxiliary network to extract more effective feature representations. The channel attention module emphasizes the interpretation of the informative part of the given input image and sees its meaning in it. The spatial attention module emphasizes the spatial location of the informative part of the input, supportive of channel attention. We have used the Convolutional Block Attention Module (CBAM) [18]. The used CBAM can be described as

$$F^c = M^c\left(F^i\right) \odot F^i, \quad (5)$$

$$F^s = M^s\left(F^c\right) \odot F^c. \quad (6)$$

Here in equations (5) and (6), $F^i \in R^{C \times H \times W}$ the input feature map, "$\odot$" refers to element-wise multiple, F^c and F^s are the output feature maps, M^c and M^s are the channel and spatial attention functions, respectively. The channel attention function $M^c(F^i)$ and spatial attention function $M^s(F^c)$ are expressed as

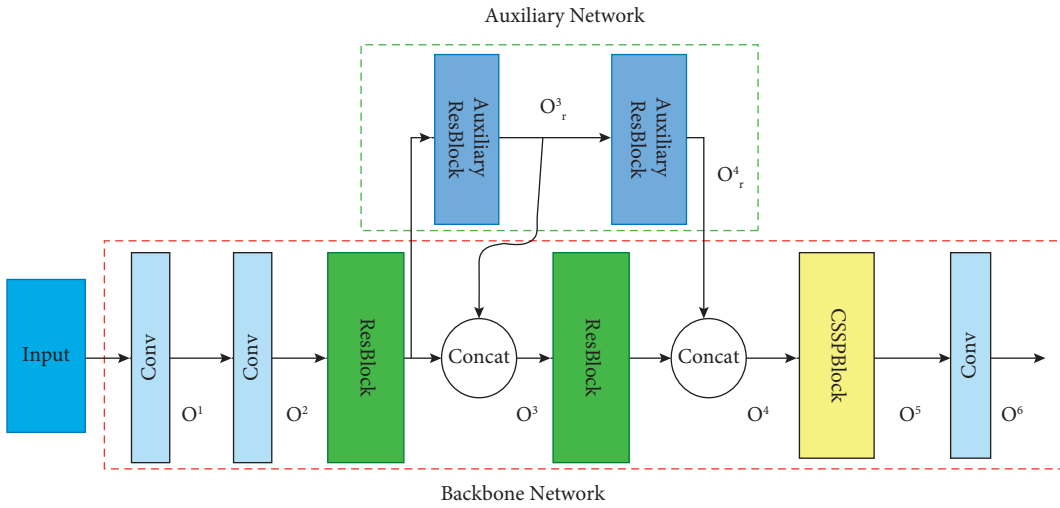


Figure 2: Proposed backbone network.

$$M^c(F^i) = \mathbf{S}\left(MLP\left(avgPooling\left(F^i\right)\right) + MLP\left(\text{maxPooling}\left(F^i\right)\right)\right)$$
$$M^s(F^c) = \mathbf{C}^{5\times5}\left[\text{maxPooling}\left(F^c\right); +\text{avgPooling}\left(F^c\right)\right]. \tag{7}$$

In equation (7), **S** is the sigmoid function, MLP () is the multilayer perceptron, and $\mathbf{C}^{7\times7}$ is the convolutional operation having a filter size of 5×5. Max Pooling and average pooling operations in spatial attention function are combined through concatenation, referred to as ":".

Figure 3 shows the proposed auxiliary network having two convolution layers to obtain the global contextual information and channel and spatial attention to get more effective information. The output representation of the first convolution layer output received from spatial attention operating is concatenated to combine both outputs, the output of the auxiliary network. Then the final output of the auxiliary network is combined with the output of the residual network of the backbone network and used as input for the next residual network there. This joining of both outputs enhances the backbone network to extract local and global information about the object and increases the accuracy of the detection network.

The architecture of the whole YOLOv4-tiny object detector is shown in Figure 4, where the proposed network is distinguished by the blue color. Compared to YOLOv4-tiny [37], the proposed object detector has replaced both CSPBlock units with two ResBlock. Moreover, the auxiliary network is also designed using two 3×3 Conv layers, a channel attention module and a spatial attention module, and a concatenation operation to obtain global information. Finally, auxiliary and backbone networks are combined to make a feature extractor.

3.2. Activity Recognition. For the activity recognition in the videos, we propose a 3D CNN where we use three-dimensional convolutions to count features in both the temporal and spatial dimensions in the later stages of CNNs. Convoluting a three-dimensional kernel to the cube obtained by assembling several spatiotemporal patches in a contiguous manner yields the 3D convolution as shown in Figure 5. The feature maps in the convolution layer are connected to multiple frames arranged consecutively in the previous layer to capture motion-related details [39]. If the kernel weights are duplicated around the patch cube, the 3D convolution kernel can only select one form of a function from the patch cuboid. The number of feature maps expands as the number of layers increases on CNN, which helps create various sorts of features from the lowest available maps.

To build the 3D cube, convolve a 3D filter kernel by stacking multiple contiguous frames. The function maps are linked to multiple adjacent frames using this operation. The working mechanism of 3d CNN is described in (1), where the value at position (a, *b*, c) in the kth feature map in the lth layer is described as

$$\mathbf{v}^{\mathbf{a,b,c}}_{\mathbf{k,l}} = \mathbf{tanh}\left(\mathbf{y_{kl}} + \sum_{\mathbf{m}}\sum_{\mathbf{x=0}}^{\mathbf{X_k-1}}\sum_{\mathbf{y=0}}^{\mathbf{Y_k-1}}\sum_{\mathbf{z=0}}^{\mathbf{Z_k-1}} \mathbf{w}^{\mathbf{xyz}}_{\mathbf{klm}}\mathbf{v}^{\mathbf{(a+x)(b+y)(c+z)}}_{\mathbf{(k-1)m}}\right). \tag{8}$$

where $\mathbf{w}^{\mathbf{xyz}}_{\mathbf{klm}}$ is the feature map linked to the m^{th} value of the kernel in the previous layer, and ZK is the 3D filter kernel size along the temporal axis. The architecture of the proposed model is shown in Figure 6.

To increase the model's efficiency, it uses 3 layers, including convolutional, pooling, and fully connected layers. In the convolutional layer, a filter layer of learned parameters converts images into processable data in this layer. Each kernel filters for a different function, and each analysis employs several kernels. In a convolution, only small parts of an image are looked at. They are assigned and transformed to an activation map that represents the image layers based on how likely it is that they belong to a certain filter class. To create three-dimensional activation maps, the kernels in a 3D CNN traverse across the three data dimensions of height, length, and depth. In pooling layers. The activation maps created during convolution are pooled or down-sampled. Pooling is similar to convolution in that it involves moving a filter around an activation map and testing a small segment simultaneously. This filter abstracts either the scanned area's

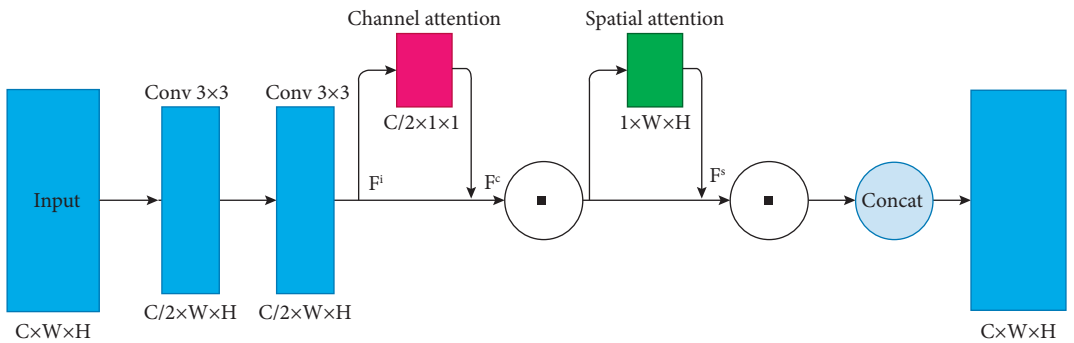


Figure 3: Auxiliary residual network.

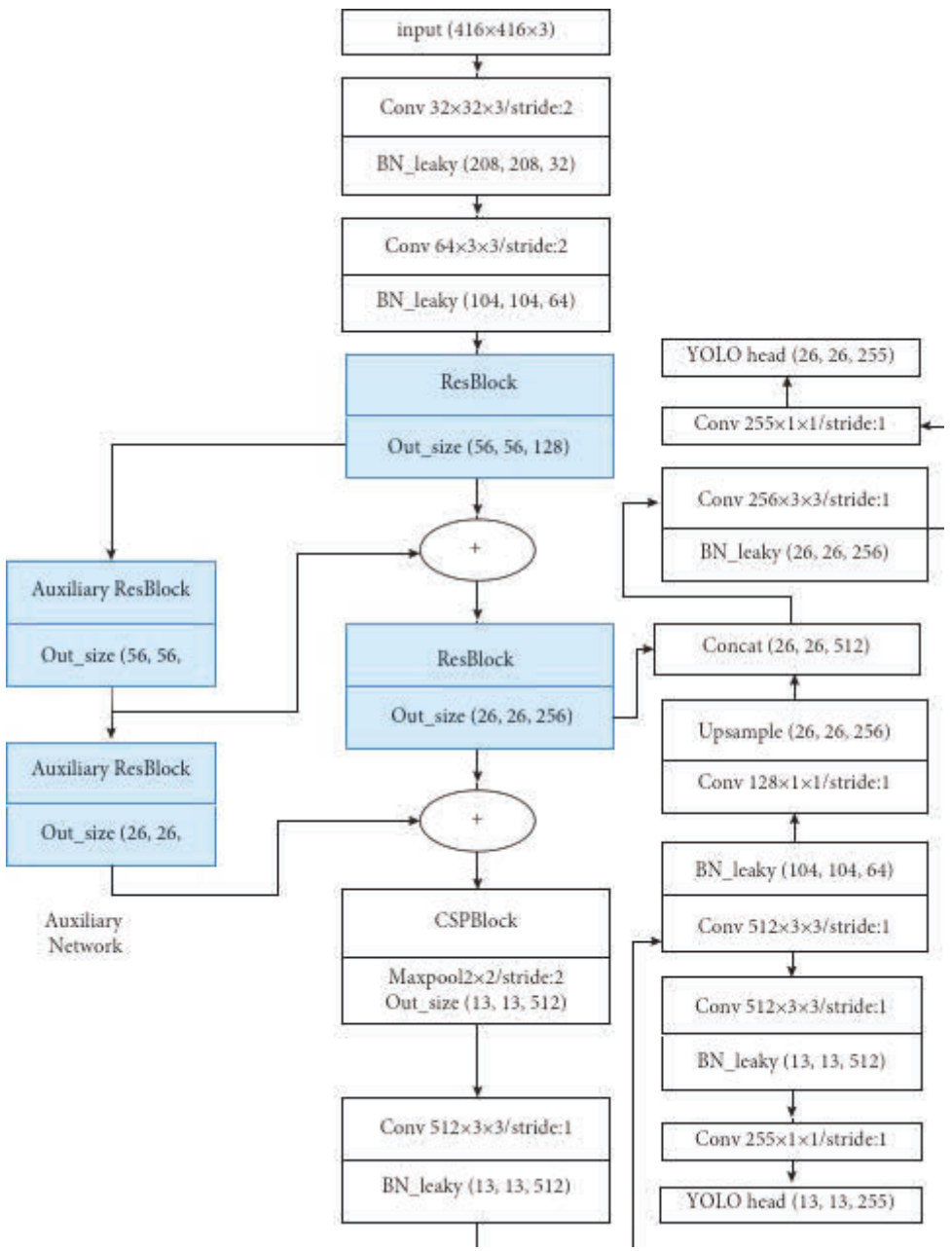


Figure 4: YOLOv4-tiny architecture with proposed changes in blue color.

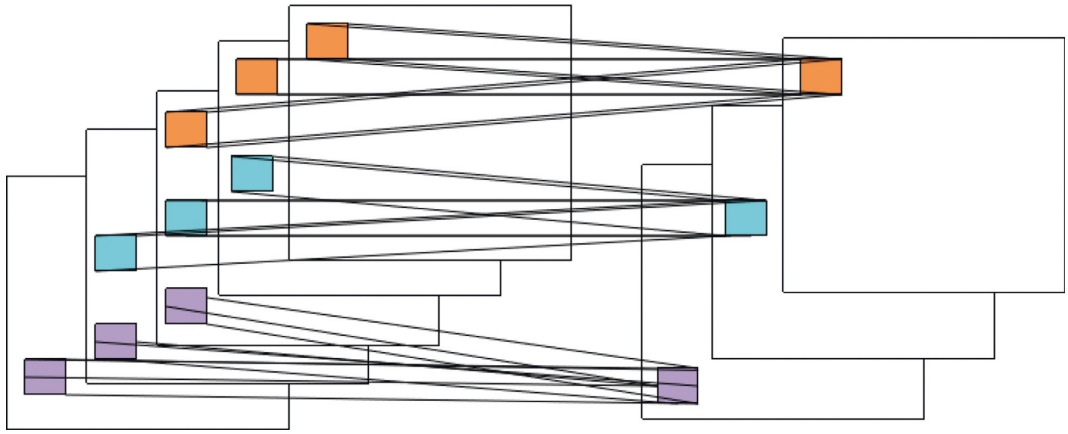


Figure 5: 3D-CNN architecture.

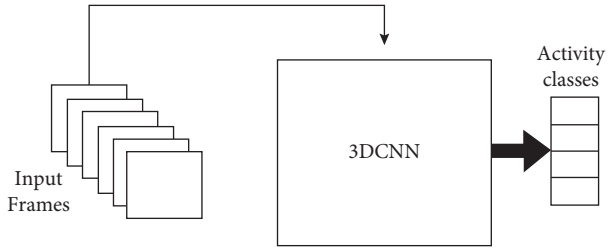


Figure 6: Flow of video recognition.

average, a weighted average dependent on the central pixel, or the extreme value of a new map.

The output layers are compressed, the probabilities found are evaluated, and the output is allotted a value, a logit, after several iterations, often thousands, of convolution and pooling. This analysis is carried out by the completely connected layer, in which each flattened output layer is interpreted by linked nodes, similar to a fully connected neural network as exhibited in Figure 6. Using hyper-parameters such as zero-padding (P), receptive field (R), stride length (S), and volume dimension (depth × width × height), calculate the spatial size of the 3D CNN output volume. We used image input of dimension $I \times J \times K$, where $I = 224$, $J = 224$, and $K = 3$ where J stands for row pixel values, I for column pixel values, and K stands for the number of channels in this work, which is three. To measure the neurons in the Convolutional layer, multiply $((W - F + 16.P)/S) + 1$. The input layer is $((224–11 + 16.0)/1) + 1 = 229$, resulting in an output volume of $229 \times 229 \times 32$, where height, width = 224 is the input frame's height and width, $F = 11 \times 11 \times 16$ is the 3D filter depth, $P = 0$ is the zero-padding, and $S = 1$ is the stride that leads to the output.

3.3. Smart Surveillance Using Internet of Things. The Internet of Things (IoT) has been utilized for efficient decision-making in real time. We utilized Ethernet communication to create the local server such that we have assigned a specific I.P. (Internet Protocol) address to access each camera feed present in the particular location. The flow of IoT architecture in the proposed architecture has been depicted in Figure 7. Ethernet provides minimal latency in the IoT environment and LAN (Local Area Network) for smooth communication of interconnected devices. Ethernet cables are not like other wires. The stream of any particular camera is then passed to the centralized GPU (Graphic Processing Unit) for processing and making predictions. All the decisions based on the predictions are then made available to the local network for efficient and quick response. We utilized the IoT concept because we could control each process remotely and monitor it as well. We can use different communications protocols on that according to system needs as well. The Ethernet communication protocols use this architecture to control feed monitoring and prediction remotely. The proposed work utilized the Local Area Network (LAN) technology that connects Internet devices using wired communication. It described how data is shared through a physical medium from one device to another network device. It is a link-layer protocol in the TCP/IP stack. It is based on the IEEE 802.3 standard [36]. In the proposed work, Ethernet is used to connect stationary or fixed IoT devices within an IoT system. Ethernet cable served as a wired medium for connecting computers, IP cameras, servers, switches, and routers.

We managed a stream of CCTV (Close Circuit Cameras) using a Network Video Recorder (NVR) and BNC (Bayonet Neill–Concelman) cable. The CCTV framework is arranged to impart its signal to an advanced video recorder, i.e., a DVR using BNC cable. The NVR contains five hard discs of 1 terabyte (TB) each for video film recording. It underpins HDMI (High-Definition Multimedia Interface) or VGA (Video Graphics Array) video yield, which permits focal observation on the LCD screen or TV. The proposed DVR includes video, live web-based streaming, and playback. An I.P surveillance camera communicates its signals alongside its network. I.P. security cameras used in the proposed system utilized a CAT-6 link to convey signals to the network video recorder (NVR). As a result, we achieved a higher resolution stream, efficient, real-time accessibility, and video/sound secured transmission using an IoT-based architecture.

In this work, cameras' live recordings are accessed through each specific I.P. address that is further processed by a centralized GPU-based server. The analysis and anomaly predictions are performed using the proposed computer vision-

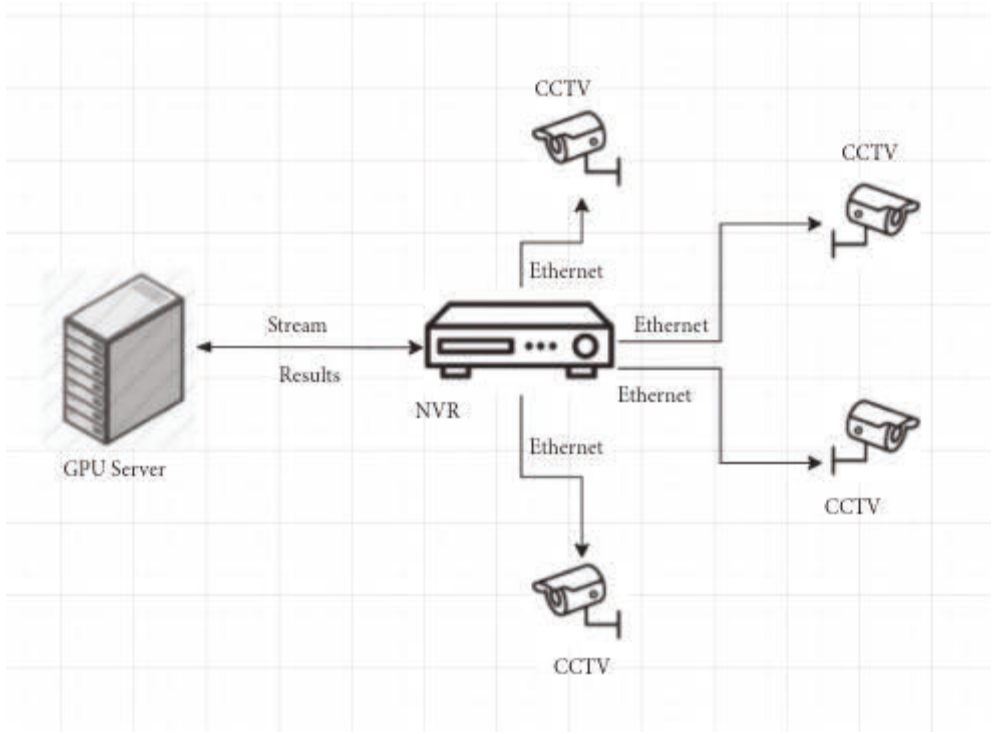


Figure 7: Internet of Things-based architecture for decision making.

based hybrid models (YOLO-v4 and 3D-CNN) to identify the specific activities in the live recordings. In case of an emergency or any suspicious activity, it can send alerts and notifications to the relevant person or authority for immediate action.

4. Experimental Results and Performance Analysis

4.1. Datasets. Benchmark datasets play a vital role in results and performance analysis in the state of the art [14]. Surveillance videos can capture a variety of real anomalies. The proposed methodology is evaluated on two major datasets. First, the UCF-Crime dataset [40] consists of various real-world anomalies, including smoke fighting, robbery, snatching, and vandalism, on which the proposed model is evaluated. These acidities are selected because they are considered prohibited. These activities are recognized based on the overall activity of the given sequence rather than the individual activities of the actors. Each mentioned activity has approximately 7000 frames in the used dataset. For training and evaluation, around 2000 frames of each activity are selected. A brief description of each activity chosen is given:

(a) Smoking: this event contains videos showing people smoking in public places such as university campuses.

(b) Fighting: this activity is based on fighting between or among people in public places such as university campuses.

(c) Snatching: this activity is based on various objects snatching, including purses, handbags, cell phones, and laptops.

(d) Gun pointing: in addition to fighting, this activity class requires gun objects in the sequence.

(e) Vandalism: this class represents a group action involving deliberate destruction of or damage to objects like buildings, vehicles, furniture, etc.

Moreover, we gathered a dataset of suspicious objects involved in those activities. We used images from the UCF crime dataset [40] and the COCO dataset [41]. Additionally, we performed preprocessing and annotation labeling on the data collected from datasets.

We have pretrained the Modified-YOLOv4, 3D-CNN network on COCO, and the whole network is fine-tuned on the CUF crime dataset, which has approximately 2000 frames for each of the five activities. The Modified-YOLOv4 is trained on the 2000 frames of each activity for 1500 epochs, while the 3D-CNN model is trained for 2000 epochs such that 80% of the dataset is devoted to training and 20% to validation. Figure 8 shows the Modified-YOLOv4 accuracy graph, from the discrepancy between training and validation accuracy. It can be deduced that the model is somewhat overfitting training data, a characteristic of deep learning models. The Modified-YOLOv4 has a training accuracy of 96.2% and a validation accuracy of 94.21%. After 1500 epochs, the training loss for Modified-YOLOv4 decreased from 8.6 to 0.19, while the validation loss decreased from 8.7 to 0.25. Figure 9 depicts the loss progression of Modified-YOLOv4 throughout training.

Figure 10 demonstrates that the 3D CNN module was similarly susceptible to overfitting since there was a discrepancy between training and validation accuracy, with validation accuracy remaining lower than training accuracy.

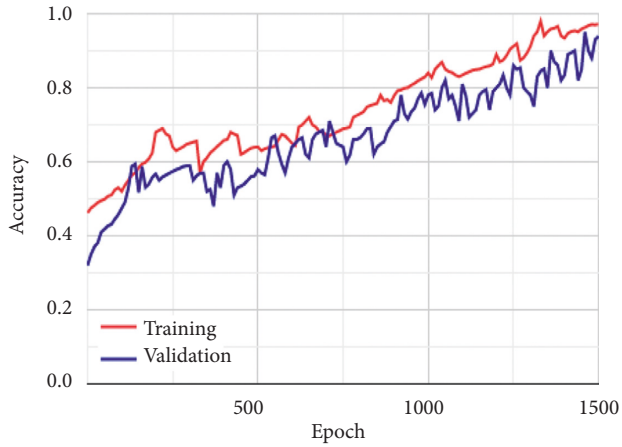


Figure 8: Accuracy graph of Modified-YOLOv4.

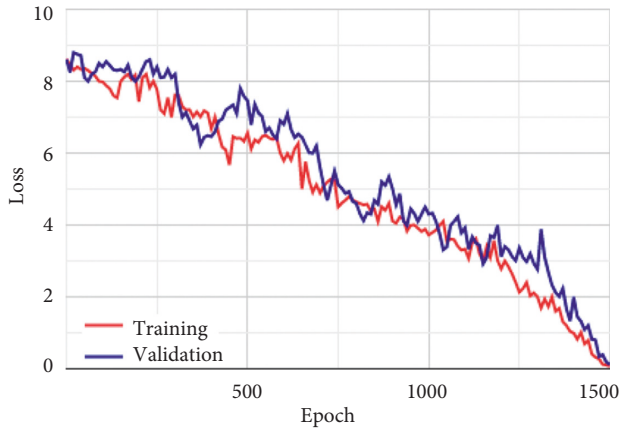


Figure 9: Loss graph of Modified-YOLOv4.

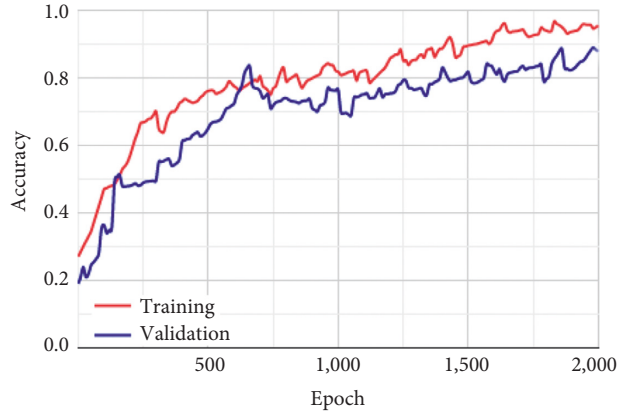


Figure 10: Training and validation accuracy graph of 3D CNN.

At the conclusion of the previous period, training accuracy was 94.8% and validation accuracy was 89.0%. Training loss (Figure 11) began decreasing from around 9.2 to 0.11, whereas validation loss began at 9.8 and finished at 0.22 in the final epoch. At some epochs, validation loss was less than training loss, but it remained significant for most of the training period.

Table 1 shows the aggregated confusion matrix of the activity recognition network. We have achieved 93.2% accuracy, 91.01% precision, and 90.1% recall. The proposed activity recognition model performed slightly poorly on the fighting and vandalism activities, while performance was highest on the gun pointing and smoking. Both precision and recall are lower because a few of the activities are

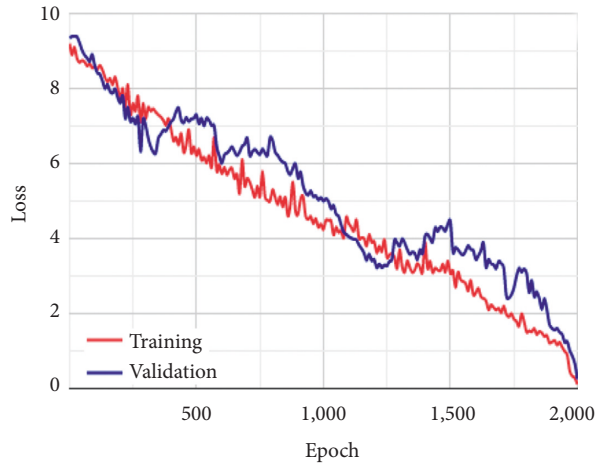


Figure 11: Training and validation Loss graph of 3D CNN.

Table 1: Confusion Matrix on proposed activities.

		Actual activity				
		Smoking	Gun pointing	Snatching	Fighting	Vandalism
Predicted activity	Smoking	48	1	1	—	—
	Gun pointing	1	49	—	—	—
	Snatching	—	1	47	2	—
	Fighting	—	—	3	44	4
	Vandalism	—	—	—	5	45

confusing. 50 activities in each category are used for this confusion matrix.

5. Conclusion

Human suspicious activity recognition is a challenging task with vast applications in video surveillance, intelligent transport systems, entertainment, and anomaly detection. Whether the event has already occurred or not, searching for the desired event through recordings is extremely time-consuming. However, a system that automatically senses any irregular or abnormal condition in advance and alerts the appropriate authorities is more appealing, and it can be used in both indoor and outdoor settings. Automatic video surveillance has tackled this issue. It is impossible to monitor CCTV events manually. Many researchers have worked on spatial information with temporal sequences for human activity recognition and detection. However, they failed to achieve impressive results in real-time. This paper presents a hybrid model which first detects the area of interest using the YOLO-v4 architecture where an anomaly or unusual activity is happening and then passes it to the 3D-CNN architecture for activity recognition based on the temporal information. The experiments performed on benchmark datasets and the 94.21% accuracy attained show the significance and robustness of the proposed architecture.

Furthermore, an IoT-based architecture has been utilized for real-time processing and efficient decision-making. The proposed multimodal is customizable, flexible, and extendable. Therefore, the system can quickly adopt new activities such as pose points, hand tracking, etc.

Data Availability

The data used in this paper are available from the corresponding author upon reasonable request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research is financially supported by Prince Mohammad bin Fahd Center for Futuristic Studies at the Prince Mohammad bin Fahd University and the World Futures Studies Federation. The authors are thankful for the support.

References

[1] T. Saba, A. Rehman, R. Latif, S. M. Fati, M. Raza, and M. Sharif, "Suspicious activity recognition using proposed deep L4-branched-ActionNet with entropy coded ant colony system optimization," *IEEE Access*, vol. 9, pp. 89181–89197, 2021.

[2] A. R. Khan, T. Saba, M. Z. Khan, S. M. Fati, and M. U. G. Khan, "Classification of human's activities from gesture recognition in live videos using deep learning," *Concurrency and Computation: Practice and Experience*, vol. 34, no. 10, Article ID e6825, 2022.

[3] T. Saba, A. Rehman, T. Sadad, H. Kolivand, and S. A. Bahaj, "Anomaly-based intrusion detection system for IoT networks

through deep learning model," *Computers & Electrical Engineering*, vol. 99, Article ID 107810, 2022.

[4] I. Imran, S. Din, G. Jeon, and G. Fortino, "Towards collaborative robotics in top view surveillance: a framework on using deep learning," *IEEE/CAA Journal of Automatica Sinica*, vol. 8, no. 7, pp. 1253–1270, 2021.

[5] M. A. Khan, H. Arshad, R. Damaševičius et al., "Human Gait Analysis: A Sequential Framework of Lightweight Deep Learning and Improved Moth-Flame Optimization Algorithm," *Computational Intelligence and Neuroscience*, vol. 2022, 2022.

[6] Y. Al-Hamar, H. Kolivand, M. Tajdini, T. Saba, and V. Ramachandran, "Enterprise credential spear-phishing attack detection," *Computers & Electrical Engineering*, vol. 94, Article ID 107363, 2021.

[7] H. Yar, T. Hussain, Z. A. Khan, D. Koundal, M. Y. Lee, and S. W. Baik, "Vision sensor-based real-time fire detection in resource-constrained IoT environments," *Computational Intelligence and Neuroscience*, vol. 2021, pp. 1–15, 2021.

[8] F. Orujov, R. Maskeliūnas, R. Damaševičius, W. Wei, and Y. Li, "Smartphone based intelligent indoor positioning using fuzzy logic," *Future Generation Computer Systems*, vol. 89, pp. 335–348, 2018.

[9] I. Abunadi, "Enterprise architecture best practices in large corporations," *Information*, vol. 10, no. 10, p. 293, 2019.

[10] B. Al, F. Orujov, R. Maskeliūnas, R. Damaševičius, and A. Venčkauskas, "Fuzzy logic type-2 based wireless indoor localization system for navigation of visually impaired people in buildings," *Sensors*, vol. 19, no. 9, p. 2114, 2019.

[11] G. Vallathan, A. John, C. Thirumalai, S. Mohan, G. Srivastava, and J. C. W. Lin, "Suspicious activity detection using deep learning in secure assisted living IoT environments," *The Journal of Supercomputing*, vol. 77, no. 4, pp. 3242–3260, 2021.

[12] M. Yousuf, Z. Mehmood, H. A. Habib, T. Mahmood, and M. Rehman, "A novel technique based on visual words fusion analysis of sparse features for effective content-based image retrieval," *Mathematical Problems in Engineering*, vol. 2018, pp. 1–13, 2018.

[13] I. M. Nasir, M. Raza, J. H. Shah, S. H. Wang, U. Tariq, and M. A. Khan, "HAREDNet: a deep learning based architecture for autonomous video surveillance by recognizing human actions," *Computers & Electrical Engineering*, vol. 99, Article ID 107805, 2022.

[14] J. L. González, C. Zaccaro, J. A. García, L. M. Morillo, and F. Caparrini, "Real-time gun detection in CCTV: an open problem," *Neural Networks*, vol. 132, pp. 297–308, 2020.

[15] C. D. J. I. Zong, "Smart security system for suspicious activity detection in volatile areas," *Journal of Information Technology and Digital World*, vol. 02, no. 01, pp. 64–72, 2020.

[16] H. Kolivand, M. S. Rahim, M. S. Sunar, A. Z. A. Fata, and C. Wren, "An integration of enhanced social force and crowd control models for high-density crowd simulation," *Neural Computing & Applications*, vol. 33, no. 11, pp. 6095–6117, 2021.

[17] M. E. Issa, A. M. Helmi, M. A. A. Al-Qaness, A. Dahou, M. A. Elaziz, and R. Damaševičius, "Human activity recognition based on embedded sensor data fusion for the internet of healthcare things," *Healthcare*, vol. 10, no. 6, p. 1084, 2022.

[18] G. Şengül, E. Ozcelik, S. Misra, R. Damaševičius, and R. Maskeliūnas, "Fusion of smartphone sensor data for classification of daily user activities," *Multimedia Tools and Applications*, vol. 80, no. 24, pp. 33527–33546, 2021.

[19] M. T. Bhatti, M. G. Khan, M. Aslam, and M. J. Fiaz, "Weapon detection in real-time CCTV videos using deep learning," *IEEE Access*, vol. 9, pp. 34366–34382, 2021.

[20] F. Afza, M. A. Khan, M. Sharif et al., "A framework of human action recognition using length control features fusion and weighted entropy-variances based feature selection," *Image and Vision Computing*, vol. 106, Article ID 104090, 2021.

[21] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea, "Machine recognition of human activities: a survey," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 18, no. 11, pp. 1473–1488, 2008.

[22] Y. ZhuZhu, N. M. Nayak, and A. K. Roy-Chowdhury, "Context-aware activity modeling using hierarchical conditional random fields," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 7, pp. 1360–1372, 2015, Jul.

[23] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici, "Beyond short snippets: deep networks for video classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4694–4702, Boston, MA, USA, June 2015.

[24] M. S. Ibrahim, S. Muralidharan, Z. Deng, A. Vahdat, and G. Mori, "A hierarchical deep temporal model for group activity recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1971–1980, Las Vegas, NV, USA, June 2016.

[25] P. Khaire, P. Kumar, and J. Imran, "Combining CNN streams of RGB-D and skeletal data for human activity recognition," *Pattern Recognition Letters*, vol. 115, pp. 107–116, 2018.

[26] M. Gnouma, A. Ladjailia, R. Ejbali, and M. Zaied, "Stacked sparse autoencoder and history of binary motion image for human activity recognition," *Multimedia Tools and Applications*, vol. 78, no. 2, pp. 2157–2179, 2019.

[27] Y. Xing, C. Lv, H. Wang, D. Cao, E. Velenis, and F. Y. Wang, "Driver activity recognition for intelligent vehicles: a deep learning approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 5379–5390, 2019.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[29] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, Boston, MA, June 2015.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, June 2016.

[31] Z. Chang, X. Ban, Q. Shen, and J. Guo, "Research on three-dimensional motion history image model and extreme learning machine for human body movement trajectory recognition," *Mathematical Problems in Engineering*, vol. 2015, pp. 1–15, 2015.

[32] X. F. Ji, Q. Q. Wu, Z. J. Ju, and Y. Y. Wang, "Study of human action recognition based on improved spatio-temporal features," *International Journal of Automation and Computing*, vol. 11, no. 5, pp. 500–509, 2015.

[33] Y. Zhong, B. Xu, G. T. Zhou, L. Bornn, and G. Mori, "Time Perception Machine: Temporal point Processes for the when, where and what of Activity Prediction," 2018, https://arxiv.org/abs/1808.04063.

[34] T. Saba, "Intrusion detection in smart city hospitals using ensemble classifiers," in *Proceedings of the 2020 13th International Conference on Developments in eSystems Engineering (DeSE)*, pp. 418–422, IEEE, Liverpool, United Kingdom, June 2020.

[35] J. Patalas-Maliszewska, D. Halikowski, and R. Damaševičius, "An automated recognition of work activity in industrial manufacturing using convolutional neural networks," *Electronics*, vol. 10, no. 23, p. 2946, 2021.

[36] D. A. John, "IEEE 802 LMSC," 2022, https://standards.ieee.org/standard/802_3-2018.html.

[37] A. Bochkovskiy, "Darknet: Open-Source Neural Networks in Python," 2021, https://github.com/AlexeyAB/darknet.

[38] A. Bochkovskiy, C. Y. Wang, and H. Y. Liao, "Yolov4: Optimal Speed and Accuracy of Object Detection," 2020, https://arxiv.org/abs/2004.10934.

[39] J. Arunnehru, G. Chamundeeswari, and S. P. Bharathi, "Human action recognition using 3D convolutional neural networks with 3D motion cuboids in surveillance videos," *Procedia Computer Science*, vol. 133, pp. 471–477, 2018.

[40] W. Sultani, C. Chen, and M. Shah, "Real-world anomaly detection in surveillance videos," 2018, https://arxiv.org/abs/1801.04264.

[41] T. Y Lin, M. Michael, B. Serge et al., "Microsoft coco: common objects in context," 2014, https://arxiv.org/abs/1405.0312.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 5363764, 12 pages
https://doi.org/10.1155/2022/5363764

Research Article

E-minBatch GraphSAGE: An Industrial Internet Attack Detection Model

Jin Lan, Jia Z. Lu, Guo G. Wan, Yuan Y. Wang, Chen Y. Huang, Shi B. Zhang, Yu Y. Huang, and Jin N. Ma

School of Cybersecurity, Chengdu University of Information Technology, Chengdu 610225, China

Correspondence should be addressed to Jia Z. Lu; ljz@cuit.edu.cn

Received 3 March 2022; Revised 30 May 2022; Accepted 16 June 2022; Published 14 July 2022

Academic Editor: Robertas Damaševičius

Copyright © 2022 Jin Lan et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The Industrial Internet has grown rapidly in recent years, and attacks against the Industrial Internet have also increased. When compared with the traditional Internet, the industrial Internet has a more complex network structure, and the traditional graph neural network attack behavior detection model cannot well adapt to the complex network environment. To make the model better adapt to the complex network environment, this paper proposes the E-minBatch GraphSAG model. First, the application layer source port and source IP address is used as source nodes, the application layer target port and target IP address are used as target nodes, and the remaining traffic information is used as edge information to complete the construction of the graph structure data, and then the constructed graph structure data is presampled to select the edge information that needs to be aggregated next, followed by using the AGG aggregation function to aggregate the information in the domain generated by the presampling process. Finally, the information of two adjacent nodes is aggregated as edge information to classify the edges. Increase the number of IP addresses in the UNSW-NB15 dataset, and then use it for model training and testing. The experimental results show that the accuracy of the model reaches 99.49% in a relatively complex network environment. In this paper, the E-minBatch GraphSAG model is presented in an attempt to solve the problem of attack detection in the complex industrial Internet environment.

1. Introduction

Because the traditional industrial production network is separated from the Internet, and the traditional industrial control protocol does not take into account the security events that may occur during use, most traditional industrial control protocols have security problems [1]. With the rapid development of Internet technology, more and more Internet technologies are used in industrial production processes to achieve the goal of automating industrial production processes and reducing production costs [2], resulting in a new concept-industrial Internet. There is also a big difference between the modern industrial Internet and the traditional Internet. The main difference between the Industrial Internet and the traditional Internet is that the traditional Internet has a close connection with people, while the Industrial Internet has a close connection with things. The architecture of the Industrial Internet is also quite different from the traditional Internet architecture. In the modern Industrial Internet, the enterprise management, the supervisory layer, and the field layer are the main components [3].

The social impact of industrial Internet security incidents is far greater than the social impact of traditional Internet security incidents. In recent years, attacks on the Industrial Internet have gradually increased. Iran's Natanz nuclear enrichment site was attacked by the Stuxnet computer virus in 2010, causing abnormal acceleration of uranium enrichment centrifuges and eventually leading to their destruction [4]. It also opened the curtain for attacks against the industrial Internet. In 2015, the malware BlackEnergy3 [5] hacked into the control center of the Ukrainian power grid and tampered with the control commands of the relays via VPN causing widespread power outages in Ukraine. BlackEnergy3 compromised the network and software of the grid control system, launching a DDoS attack that prevented

the control system from sensing abnormal system conditions, thus preventing power from being restored to the blackout area for a long time. During Black Hat 2017, Dr. Staggsp [6] demonstrated how to hack into a wind farm's control system by physically connecting to an uncontrolled wind turbine in the United States. In 2021, a state of emergency was declared in the United States after the hacker group "DarkSide" attacked the largest fuel pipeline operator in the country [7]. Several security incidents have shown that the industrial Internet faces huge security risks, and artificial intelligence-based attack detection systems can help to provide early warning of attacks and greatly improve the security of the system.

Detecting attacks is a key step in securing the industrial Internet, and alerting to attacks as early as possible can reduce the impact of attacks to a manageable extent. Currently, there are different classification results for different intrusion detection systems based on the classification method [8]. There are two types of data source classification: host-based and network-based. Classification based on the detection technique can be classified as misuse-based approach and anomaly-based approach. Anomaly-based methods, which are currently the mainstream detection methods, can also be classified as statistical analysis-based methods [9], cluster analysis-based methods [10], artificial neural network-based methods [11], or deep learning-based methods [12]. Among them, current studies generally agree that deep learning-based methods for attack detection are more effective than the others [13]. This is because deep learning-based models have better self-learning, self-adaptive capabilities, better generalization ability, and the ability to detect unknown attack behaviors better.

Most attack behavior detection methods focus on finding attack behaviors from the attack traffic itself, while ignoring the correlation between attack traffic. In this paper, we attempt to introduce graph neural networks, a relatively new subfield in the field of deep neural network research, into attack behavior detection in the Industrial Internet.

The scale of the Industrial Internet has begun to grow explosively, and the network structure has become increasingly complex. In order to detect attacks in a complex network environment, this paper proposes an improved method based on E-GraphSAGE algorithm [14], E-GraphSAGE is a variant of GraphSAGE [15], which allows to collect graph edge information and support edge features Perform edge classification to detect malicious network flows. In this paper, the E-minBatch GraphSAGE algorithm is proposed to be able to better adapt to the complex network environments.

The contributions of this paper are mainly as follows.

(i) In this paper, we propose a new GNN model based on E-GraphSAGE, which uses information such as traffic duration and packet size as edge features of the graph, and presamples the points in the graph structure data so that the model can better adapt to the complex network environment.

(ii) This paper applies the new proposed model to industrial Internet attack detection and demonstrates the superiority of the new model by comparing it with traditional machine learning algorithms and deep learning algorithms through experiments.

(iii) The E-minBatch GraphSAGE algorithm proposed in this paper has better results in the detection of three kinds of attacks, namely Shellcode, Reconnaissance, and Exploits.

The rest of this paper is organized as follows. Section 2 discusses related work on industrial Internet attack behavior detection, Section 3 briefly introduces the basics related to GNN and GraphSAGE, Section 4 presents our new GNN model based on GraphSAGE, Section 5 gives experimental results and analysis, and Section 6 summarizes the full paper.

2. Related Works

At present, traditional machine learning or deep learning is mainly used for industrial Internet attack behavior detection. In contrast, there are relatively few researches on attack behavior detection based on graph neural network.

2.1. Traditional Industrial Internet Attack Behavior Detection Algorithms. The label-based attack behavior detection system can accurately detect the known attack behavior, but it is powerless to detect the unknown attack behavior. At the same time, the anomaly-based attack behavior detection system can effectively detect unknown attack behaviors, but an unavoidable problem is: no matter whether the attack behavior is known or not, the anomaly-based attack behavior detection system will have a large false negative rate and false positive rate. In order to enable the model to detect both unknown attack behaviors and known attack behaviors, researchers began to try to combine the two attack behavior detection systems. Khraisat et al. [16] combined the C5 classifier and a class of support vector machine classifiers to design a hybrid intrusion detection system (HIDS) that integrated the advantages of the label-based attack behavior detection system and the anomaly-based attack behavior detection system. The experimental results show that the method has a high accuracy in detecting attack data on the Bot-IoT dataset.

The traffic of attack behavior of the Industrial Internet presents the characteristics of low frequency and multistage. Li et al. [17] designed a bidirectional long-term and short-term storage network with multiple features, and the sequence feature layer and stage feature layer were introduced into the model. The model in the training phase can learn the corresponding attack range from historical data, and effectively detect attacks in different ranges. Suzen et al. [18] proposed a hybrid Deep Belief Network (DBN) attack behavior detection model. Hidden layers are updated via Contrastive Divergence (CD). Experiments show that the hybrid deep belief network model has achieved good accuracy in the detection of industrial Internet attack behavior. A multifeatured data clustering optimization model was used by Liang et al. [19] as the basis of an industrial network intrusion detection algorithm, which classifies the weighted distance and safety factor of the data according to the priority thresholds of the data attribute features of the nodes

in the data. Cluster centers are selected by choosing a node with a high safety factor, and data from around the node is matched into a cluster. In comparison with other algorithms, the experimental results demonstrate that the proposed algorithm has significant advantages in terms of detection rate and processing time. Huang et al. [20] proposed a data-driven intrusion detection method based on time-domain and frequency-domain analysis. The proposed method uses closed-loop controlled sensors, does not consume additional system resources and relies on system models, extracts time-domain and frequency-domain features, uses feature vectors under normal working conditions to build a hidden Markov model, and converts the trained hidden Markov model.

The traffic in the Industrial Internet is very complex and includes not only production networks but also other office networks. About solving the problem of massive data attack behavior detection in hybrid networks, Zhang et al. [21] proposed a data mining algorithm for massive intrusion cluster computing in hybrid networks with feature extraction under specific constraints. Multicomponent cross-detection methods are used to collect information on mixed network massive intrusions and construct models of mixed network massive intrusion signals. Regarding the intrusion interference under the constraint of fixed time-frequency window, Zhang adopts the cascade trap method to deal with it, so as to extract the localized basic volume and main function from a large amount of interference information, and obtain the complete energy distribution spectrum on the time-frequency plane. Data mining for clustering calculations with massive intrusion interference constraints is achieved with the help of the energy distribution spectrum as a guiding function.

The rapid development of the Industrial Internet has led to IoT devices widely deployed, and at the same time, attacks against IoT devices have also appeared in large numbers. IoT devices are ideal springboards for DoS attacks—low security and large numbers make IoT devices the target of many botnets. The attack behavior detection system needs to identify the nodes attacked by DoS in time, and takes measures such as isolation of the infected nodes to ensure the security of the entire industrial Internet environment. Alharbi et al. [22] proposed a Local Global Optimal Bat Algorithm (LGBA-NN) for Neural Networks to select feature subsets and hyperparameters to effectively detect botnet attacks. Experimental results show that LGBA-NN outperforms other variants in detection of multiple botnet attacks. Ali et al. [23] trained on intrusion data, features, and suspicious activity datasets. The data is trained according to different layers of the long- and short-term network to improve the accuracy of attack detection. With the help of training information, the test details are classified by extracting features and forming a sparse matrix construction. In experiments, the model's accuracy reached 99.29%.

The computing power of industrial Internet nodes is relatively poor, and the resources required for the training and deployment of attack detection models are huge. About how to reduce the resources consumed by the deployment node, Wozniak et al. [24] used RNN-LSTM classifier and NAdam optimization algorithm to build the model. Experimental results indicate that the model requires very few resources on deployment nodes.

All the above algorithms have achieved desirable performance in industrial Internet attack detection, but they all only consider the characteristics of the traffic itself or the spatial characteristics of the traffic, and do not consider the correlation between the traffic.

2.2. Industrial Internet Attack Behavior Detection Algorithm Based on Graph Neural Network. Graph neural networks are developing rapidly, and good progress has been made in their applications in many fields. However, the application of graph neural network to network attack behavior detection is still a relatively new field and deserves further research.

Lo et al. [14] proposed a model named E-GraphSAGE based on the GraphSAGE model, which supports edge classification. Taking IP addresses and application-layer ports as nodes, the data flows communicated between hosts are treated as side information, thereby classifying network flows into benign flows and attack flows. According to the experimental comparison, the model proposed by the author is generally better than the traditional attack behavior detection model. However, experiments have shown that with the increase of network complexity, the accuracy of E-GraphSAGE begins to decrease. Our method proposes an improved model based on E-GraphSAGE, which can better adapt to complex network environments.

3. Background

3.1. Industrial Internet Infrastructure. The industrial Internet attack behavior detection model is one of the methods to protect the safety of industrial production equipment and personnel. There are mainly three layers in modern industrial Internet architecture: the enterprise management layer, the supervision layer and the field layer. The enterprise management layer relies on the Internet to enable real-time monitoring and management of industrial processes and assist enterprises in making informed decisions. In addition to collecting data and transmitting it between the enterprise management layer and the field layer, the monitoring layer controls the field devices with specific logic. In the field layer, field information is perceived by the field devices, and data is exchanged between field devices via the field bus. The modern Industrial Internet architecture is shown in Figure 1.

As shown in Figure 1, industrial Internet attack behavior detection systems are generally deployed between the management and the management level of an enterprise, and between the management and the field level control level [3]. There are various attack behavior detection systems, and this paper focuses on GraphSAGE algorithm based on graph neural network.

3.2. Graphical Neural Network. Different attack detection algorithms require different input structures. The input data structure of the CNN-based attack behavior detection algorithm is the grayscale graph corresponding to the traffic. The input data structure of GNN-based attack behavior

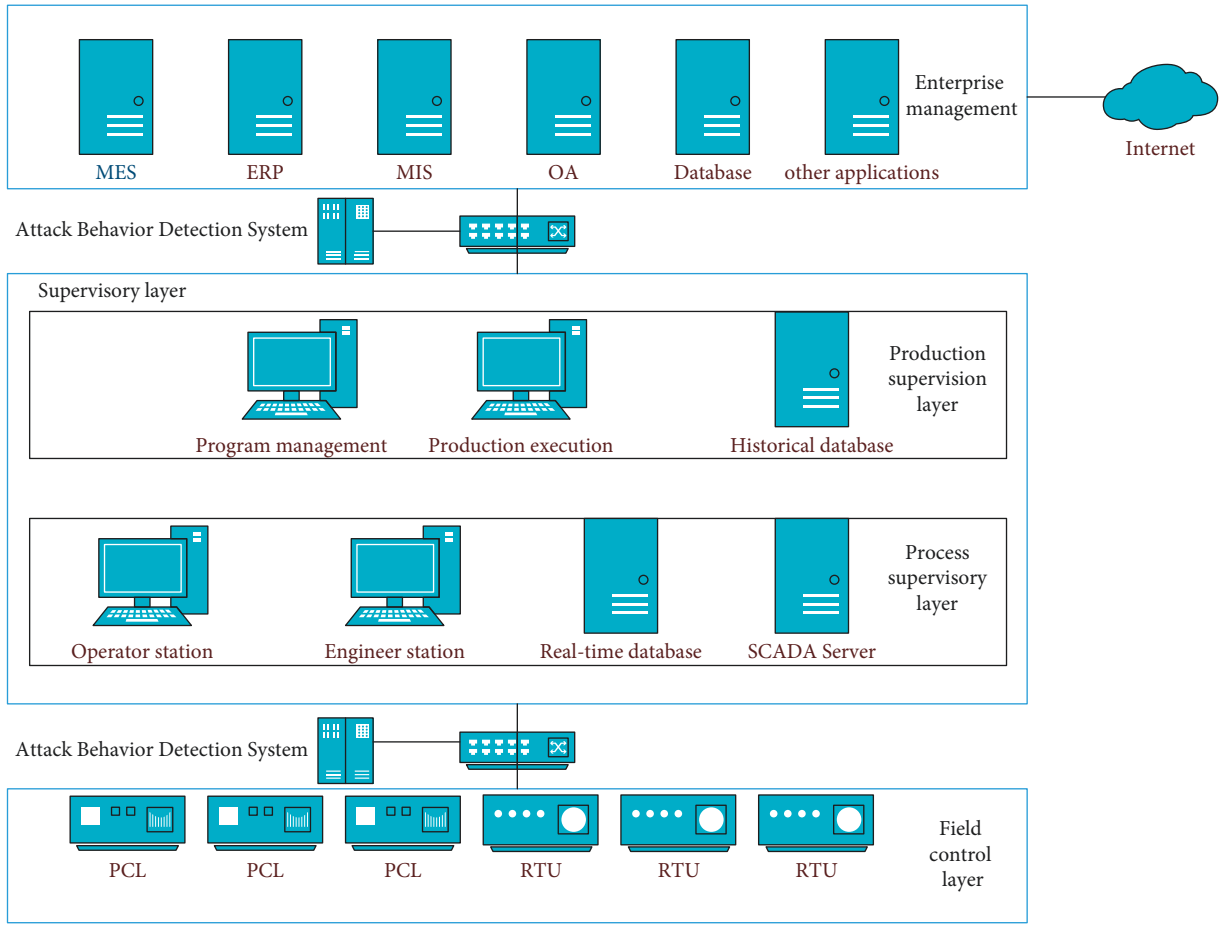


Figure 1: Industrial internet infrastructure.

detection algorithm is the IP address and application layer port as nodes, and the data flow of communication between hosts is treated as edge information, as shown in Figure 2.

Because graph neural networks can utilize data with graphical structure encountered in real-world applications (biology, telecommunications, chemistry, etc.), graph neural networks have received widespread attention since their introduction, and they have grown rapidly in recent years to become one of the fastest growing subfields of artificial intelligence.

The main reason for using GNNs for industrial Internet attack detection is that GNNs can easily exploit important structural information in network data streams. The information in network data streams can be directly encoded into a graphical format. In fact, converting network data traffic into graphical format is a method that has been used earlier, but the process is usually tedious and heavily dependent on manual labor.

3.3. GraphSAGE. GNNs can be considered as a generalization of convolutional neural networks to non-Euclidean data structures [25]. Graph neural networks use the concept of message passing to implement a generalization of the capabilities of convolutional neural networks to the processing of data with non-Euclidean structures. The messages

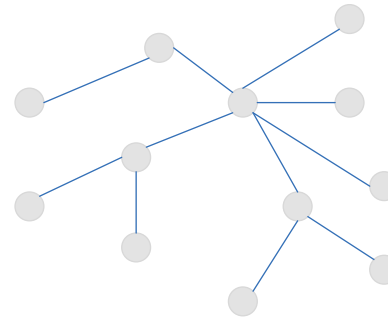


Figure 2: Figure structure.

received by a node are the result of the properties (or attributes) of the neighboring nodes of that node being aggregated. Iteration of the above process is repeated to pass the information from one node to the whole network. If in each iteration, an attempt is made to aggregate all neighboring nodes, unpredictable memory consumption and computational resource requirements occur.

Figure 3(a) shows a simple graph structure data and Figure 3(b) shows two GraphSAGE message passes to the graph. In this example, we assume that the nodes sample all neighboring nodes, i.e., information from all domain nodes

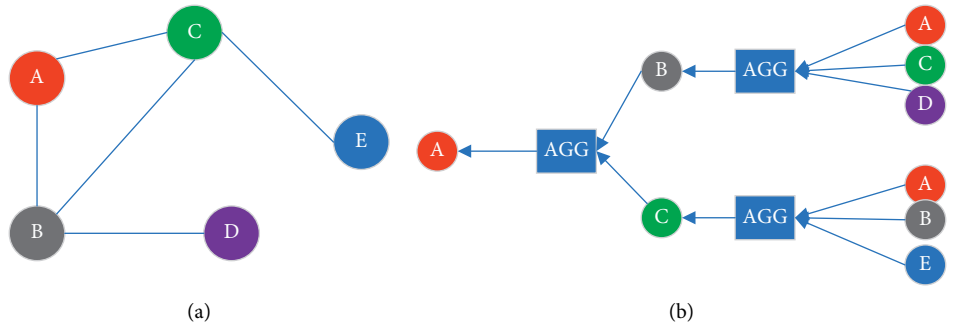


Figure 3: A given graph structure data and the corresponding two-layer fully sampled GraphSAGE algorithm model.

is considered in each iteration. In the face of more complex graph structures, sampling all nodes makes the training time and effectiveness of the model not optimized, so an attempt is made to presample the nodes [26].

Some of the symbols in the graph neural network are defined as follows: $G(v, \varepsilon)$ denotes the data of a graph structure, v is the set of points, and ε is the set of edges. The feature vector of node v is denoted as a vector X_v, and the full set of node feature vectors can be denoted as $\{X_v, \forall v \in v\}$.

In the GraphSAGE algorithm, one of the most critical hyperparameters is the number of convolutional layers K. The role of this hyperparameter is to specify the information of the algorithm's aggregated K-layer neighbor nodes. Considering both the experimental effect and the model complexity, we generally set the number of layers to $K = 2$ in the actual experimental process [26]. On the other hand, GraphSAGE needs to choose a differentiable aggregator function that aggregates the information from the neighboring nodes.

The GraphSAGE algorithm has been used in many fields with good results. However, the algorithm focuses on node classification and does not consider the problem of edge classification. The E-GraphSAGE algorithm proposed by Lo successfully solves the problem of edge classification, but cannot solve the problem of classification in complex network environment architectures. Based on the E-GraphSAGE algorithm, a new node presampling algorithm is proposed to enable the model to better detect attack behaviors in complex networks.

4. E-minBatch GraphSAGE

E-minBatch GraphSAGE is presented in this section, along with its application to detecting industrial Internet attacks.

4.1. E-minBatch GraphSAGE

4.1.1. Forward Propagation Stage. The E-GraphSAGE algorithm, compared with the traditional GraphSAGE, considers not only the node features but also the edge features, while E-GraphSAGE proposes edge embedding. The nodes are presampled in advance so that the E-minBatch GraphSAGE algorithm can adapt to complex network structures, as shown in Algorithm 1.

In comparison to E-GraphSAGE, the algorithm presented in this paper has a larger number of input nodes, which can better represent the complex network environment, and in the face of complex network structure this paper presamples the nodes once to improve the ability of attack behavior detection model to detect attack behavior in complex network environment. As shown in line 1 to 5 of the algorithm, we determine whether a node is a neighbor node of the current node, and if it is, it is directly added to the sampling range. graphSAGE recommends the use of two layers of convolution for the model, and the product of the number of neighbor nodes sampled twice is not greater than 500. The number of samples sampled twice for the model used in this paper is $S1 = 20$, $S2 = 25$(Note: $s1$ indicates that the first layer samples 20 neighbor nodes, and $s2$ indicates that the second layer samples 25 neighbor nodes). As with the E-GrapghSAGE algorithm, this paper still uses the $x_v = (1, ..., 1)$ initialized node features to aggregate the domain edges at the Kth layer.

In the aggregation function in line 9, the difference between E-minBatch GraphSAGE and GrapghSAGE algorithm is that the aggregation is not the information of surrounding adjacent nodes, but the aggregation of surrounding edge information.

$$h^k_{N(v)} = AGG_k\left\{\left\{h^{k-1}_{uv}, \forall u \in N(v), uv \in \varepsilon\right\}\right\}, \tag{1}$$

h^{k-1}_{uv} denotes $N(v)$ the edges in the sampled domain of node u in the k-1 layer and uv denotes the edge $\{\forall u \in N(v), uv \in \varepsilon\}$, $N(v)$ in the sampled domain of node v.

The calculation process in line 10 is the same as the traditional GrapghSAGE algorithm, but the calculation includes the edge information of the previous layer.

Line 11 calculates the node embedding of the kth layer, and the edge embedding Z_{uv} of the nodes in the last layer is the splicing Z_u with Z_v the node embedding, as shown in the following equation:

$$Z_{uv} = \text{CONCAT}\left(Z^K_u, Z^K_v\right), uv \in \varepsilon. \tag{2}$$

4.1.2. Back Propagation. In the back propagation phase, the method used in this paper is updated in the same way as the traditional GraphSAGE algorithm.

Input: Graph $G(v, \varepsilon)$; input edge features $\{e_{uv}, \forall_{uv} \in \varepsilon\}$; input node features $x_v = \{1, .., 1\}$, $x_v \in B$; depth K; weight matrices $W^k, \forall \in \{1, .., K\}$; non-linearity σ; differentiable aggregator functions AGG_K;
Output:Edge emdeddings $Z_{uv}, \forall_{uv} \in \varepsilon$

(1) $B^K \leftarrow B$
(2) **for** $k = K \ldots 1$ do
(3) $\quad B^{k-1} \leftarrow B^k$;
(4) $\quad$ **for** $u \in B^k$ do
(5) $\quad\quad B^{k-1} \leftarrow B^{k-1} \cup N_k(u)$;
(6) $\quad$ **end for**
(7) **end for**
(8) $h_v^0 = x_v, \forall_v \in V$
(9) **for** $k \leftarrow 1$to$K$ do
(10) $\quad$ **for** $u \in B^k$ do
(11) $\quad\quad h_N^k(v) \leftarrow \mathrm{AGG}_k(\{h_u v^k - 1, \forall u \in N(v), uv \in \varepsilon\})$
(12) $\quad\quad h_v^k \leftarrow \sigma(W^k.\mathrm{CONCAT}(h_v^k - 1, h_N(v)^k))$
(13) $\quad$ **end for**
(14) **end for**
(15) $Z_v = h_v^K$
(16) **for** $k \leftarrow 1$to$K$**do**
(17) $\quad z_{u_v^k} \leftarrow \mathrm{CONCAT}(z_u^K, z_v^K)$
(18) **end for**
(19) $z_{uv=} z_{u_v^k}$# k represents the last layer of the model#

Algorithm 1: E-minBatch GraphSAGE edge embedding.

4.2. E-minBatch GraphSAGE Attack Detection Model. As shown in Figure 4, the E-minBatch GraphSAGE attack detection model proposed in this paper first generates a network graph using network stream data, and then presamples the nodes once. After completing the presampling, the data is fed into the model for training. Finally, edge embeddings are created and classification operations are performed on the edges. The next steps are described in turn.

4.2.1. Network Diagram Construction. Network data streams are the fundamental form of data transmission in today's industrial Internet. It is also the most commonly used data format for attack detection models. The data stream contains not only the source and target of the data information, but also the size, duration, and other information of the data stream. In some scenarios, the flow is presented in the form of a graph.

There are different options for using graphs to represent data flows in different usage scenarios. In this paper, the source IP address and application layer port are used to identify the source node, and the target IP and target application layer port are used to identify the target node. The rest of the information is used as information about the edges between the source and target nodes.

Make training data and test data better represent complex network structures, and the original source IP addresses are mapped to random addresses in the range of 10.0.0.0–10.255.255.255 in this paper. A large number of IP addresses can represent the complex network more accurately and make the trained model better adapted to the complex network.

4.2.2. Presampling. In order to adapt to complex network structures, the nodes in the graph continue to be presampled after the conversion of the traffic to graph structure type is completed. In this paper, we use a two-layer convolution process, so each node is presampled twice, the first layer presamples the 20 neighbor nodes of the current node, and the second layer presamples the 25 neighbor nodes of the current node. When the number of neighboring nodes of a node cannot meet the presampling requirement, some of the neighboring nodes are sampled again.

4.2.3. Model Training. The training of the GraphSAGE model generally samples two layers of convolution [27], and similarly the E-minBatch GraphSAGE proposed in this paper uses two layers of convolution. For the aggregation function AGG, the mean value of each edge embedding is simply found, and the defined form is shown in the following equation:

$$h_{N(v)}^k = \sum_{\substack{u \in N(v) \\ uv \in \varepsilon}} \frac{h_{uv}^{k-1}}{|N(v)|_e}. \tag{3}$$

h_{uv}^{k-1} denotes the embedding of the model at layer k-1 and $|N(v)|_e$ denotes the number of aggregated neighbor nodes. In the two-layer convolution, the number of sampling neighbor nodes is $S1 = 20$, $S2 = 25$.

The size of the hidden layer as shown in (3) is set to 128 hidden units, and the nonlinear activation function is chosen as the ReLu function. For improving the model's ability to generalize, a dropout mechanism of 0.2 is set between the

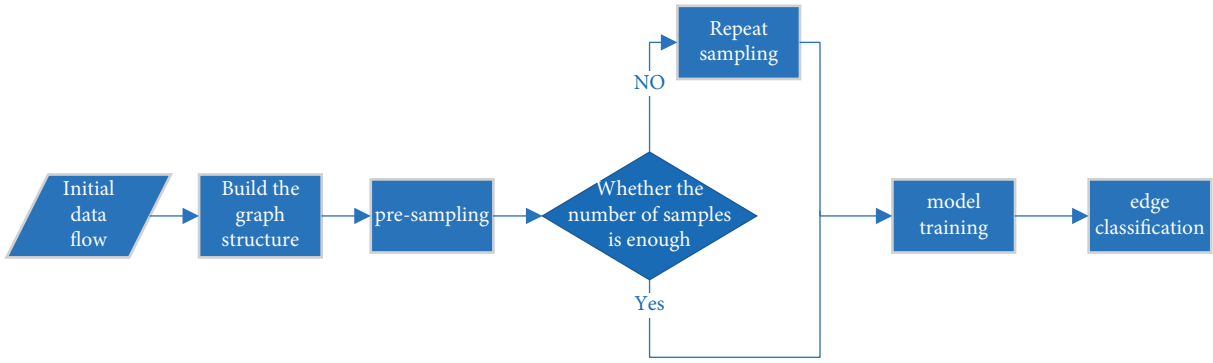


Figure 4: E-minBatch graph SAGE attack behavior detection model flow.

two convolutional layers. When generating the calculation results in the last layer, the embedding of the two nodes is spliced together to get the corresponding edge embedding, and the size of the edge embedding is 256-dimensional at this time, and the edge embedding is passed through a Log Softmax layer, which facilitates the training and optimization of the model parameters.

4.2.4. Edge Classification. After the model training is completed, the effectiveness of the E-minBatch GraphSAGE model is evaluated using the test set. The test set also needs to be transformed into a graph structure as well, presampled, passed through the trained E-minBatch GraphSAGE layer, and finally passed through the Log Softmax layer edge corresponding to the probabilities of different classes, and finally compared with the real class labels to calculate the classification evaluation performance metrics.

5. Dataset and Experimental Results

In this section, this paper presents the datasets selected for training and testing along with the evaluation criteria of the experiments, and finally the experimental results of the model.

5.1. Dataset. The model was pretrained with the UNSW-NB15 [28] dataset, which was generated by the IXIA PerfectStorm tool from the Australian Cyber Security Centre (ACCS) Cyber Scope Lab. The number of various types of traffic included in UNSW-NB15 is shown in Table 1.

5.2. Evaluation Criteria. In this paper, the parameters shown in Table 2 are used to evaluate the selected model and the model proposed in this paper.

In the experiments, two labels are defined for UNSW-NB15, one indicating whether the traffic is attack traffic, and if it is, the other label what kind of attack traffic the traffic is. The first label is used for dichotomous classification and the second label is used for multiclassification. In our experiments, 70% of the traffic data of the UNSW-NB15 dataset is used as the training set, and 30% of the traffic data is used as the test set.

5.3. Experimental Results. Firstly, we compare the accuracy of different models under different training times, as shown in Figure 5.

As we can see from Figure 5, the convergence speed of the graph neural network algorithm is much slower compared to the speed of other traditional neural networks. Because in graph neural networks, along with the increase in the number of network layers, information from more distant nodes needs to be aggregated, which is the reason why using the GraphSAGE algorithm suggests setting the model within two layers. The reason for the slower convergence speed of the E-minBatch GraphSAGE algorithm compared to the E-GraphSAGE algorithm is that the nodes are presampled and require more training times to aggregate the information of surrounding neighboring nodes.

The models E-GraphSAGE [14], CNN [29], RF [27], ResNet50 [30], and the model proposed in this paper are compared in terms of F1-score, ACC, Precision, and Recall.

In a complex network environment, the model proposed in this paper, as shown in Figure 6(a), *F*1score reaches 99.88%, as shown in Figure 6(b), ACC reaches 99.49%, as shown in Figure 6(c), Precision reaches 99.67%, as shown in Figure 6(d), and Recall reaches 99.74%, which is better than E-GraphSAGE. At the same time, the model proposed in this paper is slightly inferior to the current state-of-the-art deep learning model in terms of *F*1-score, ACC, Precision, and Recall, but is currently based on graph neural networks. The research on the network attack behavior detection algorithm is still in the initial stage, and there is room for further research in the future. When the E-GraphSAGE algorithm is used to detect attack behaviors, it not only considers the characteristics of the traffic itself, but also considers the correlation between the traffic. Therefore, in a complex network environment, the effect of the E-GraphSAGE algorithm will decline to a certain extent. The purpose of E-minBatch GraphSAGE proposed in this paper is to make the attack behavior detection method based on graph neural network still has good performance in complex network environment. In the following comparative experiments, the E-minBatch GraphSAGE algorithm proposed by us and the E-GraphSAGE algorithm proposed by Lo are compared.

Table 1: UNSW-NB15 flow type, quantity and profile.

Flow type	Quantity	Introduction
Normal	2,218,761	Normal data traffic
Fuzzers	24,246	Send randomly generated fuzzy data to the target to cause the target to error into a pause state
Analysis	2,677	Port scanning, spam, and html file infiltration
Exploits	44,525	Attacks that exploit vulnerabilities known to exist in the system or software
Worms	174	Attack initiators such as viruses replicate themselves and try to infect other hosts on the network
Shellcode	1,511	A piece of code that exploits a software vulnerability
DoS	16,353	Launch a flooding attack on the target so that it cannot accept new requests
Generic	215,481	Attack against any type of group password
Reconnaissance	13,987	Simulation of information-gathering attacks
Backdoor	2,329	Bypass system defense mechanisms to access sensitive locations and sensitive information

Table 2: Model performance metrics.

Metric	Definition
Recall	$TP/TP + FN$
Precision	$TP/TP + FP$
$F1$-score	$2 \times \text{Recall} \times \text{Precision}/\text{Recall} + \text{Precision}$
Accuracy	$TP + TN/TP + FP + TN + FN$

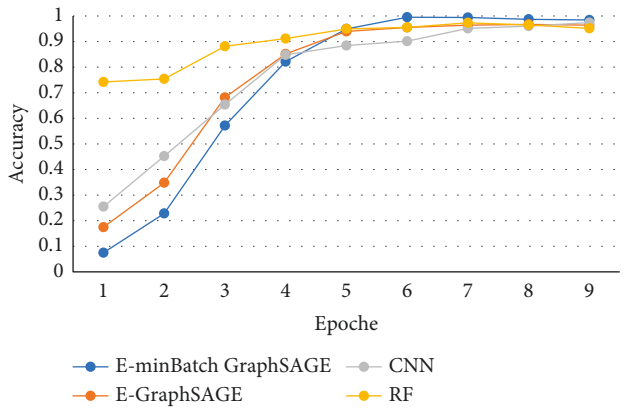


Figure 5: 10 epoche training accuracy.

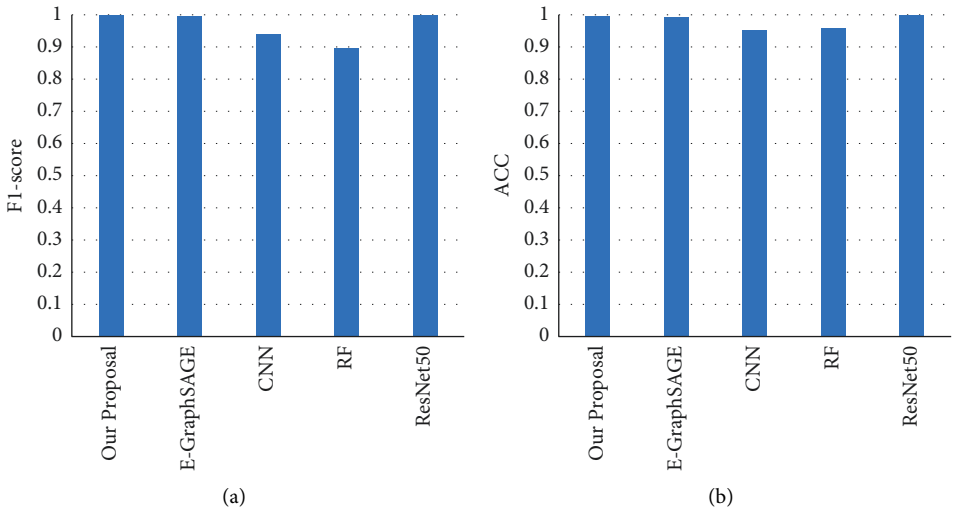


Figure 6: Continued.

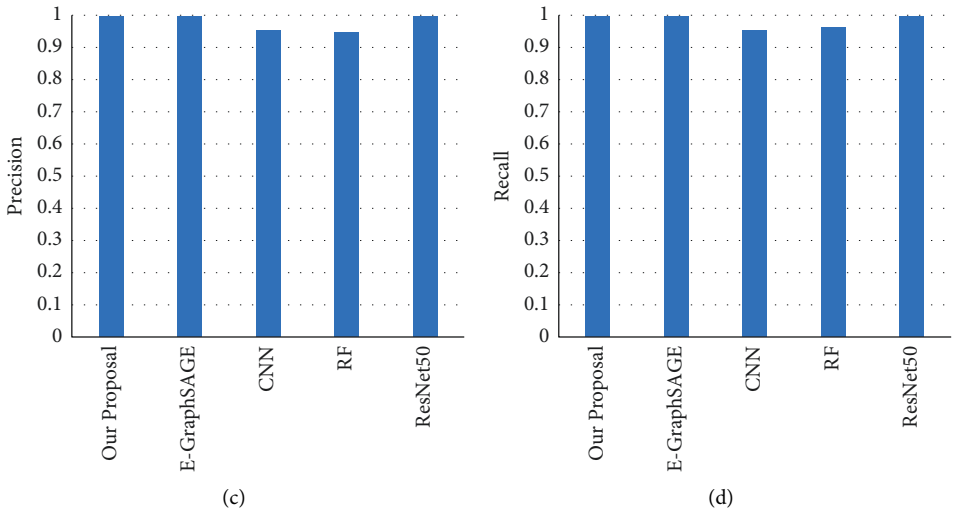


Figure 6: Compare the model proposed in this paper with E-graph SAGE, CNN, RF, ResNet50 in ACC, *F*1-score, precision, and recall. (a) *F*1-score comparison chart. (b) ACC comparison chart. (c) Precision comparison chart. (d) Recall comparison chart.

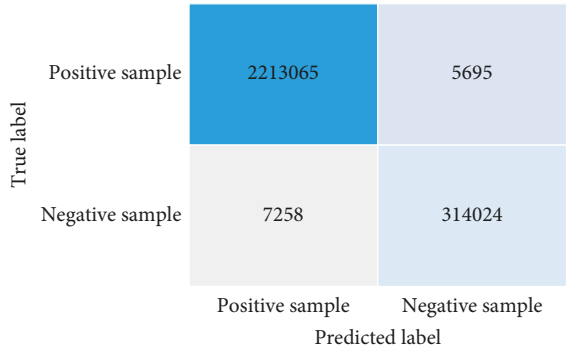


Figure 7: Compare the base confusion matrix for UNSW-NB15 dataset.

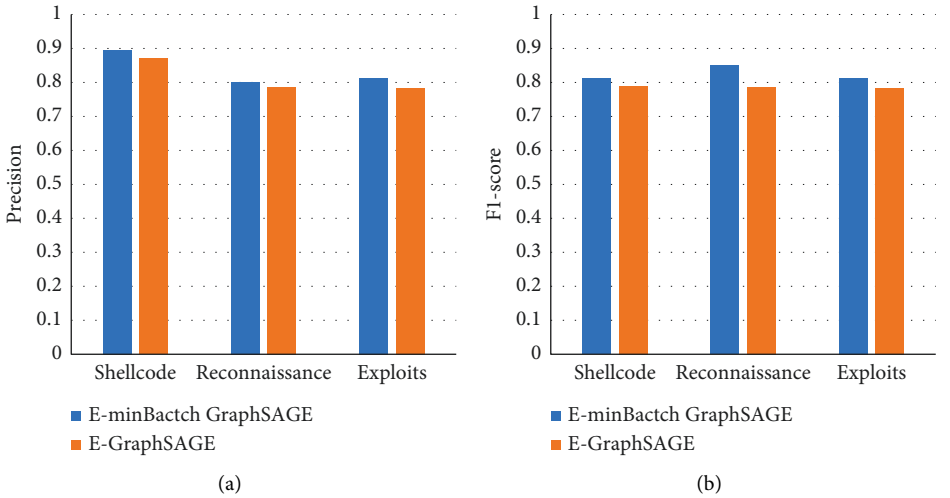


Figure 8: Continued.

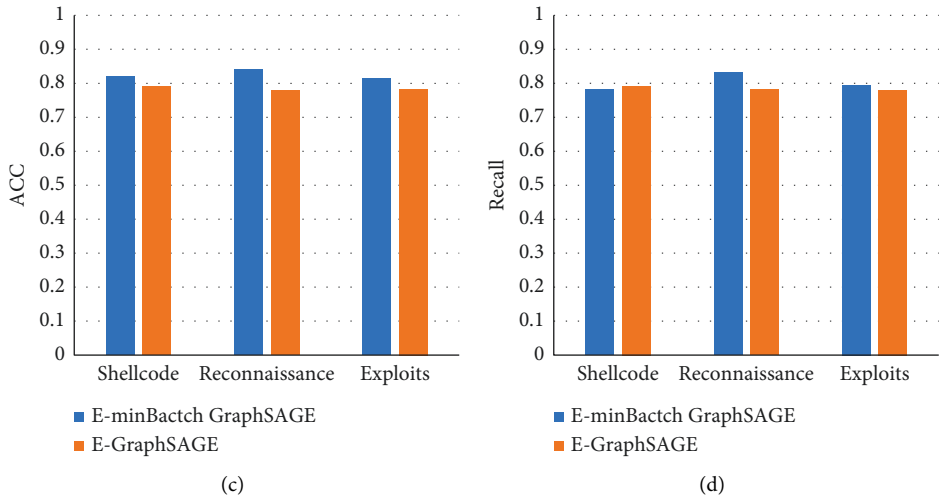


Figure 8: Compare the ACC, *F*1-score, precision, and recall of the model proposed in this paper with E-graph SAGE on three attacks: Shellcode, reconnaissance, and exploits. (a) Precision of some aggressive behaviors-1. (b) *F*1-score of some aggressive behaviors-1. (c) ACC of some aggressive behaviors-1. (d) Recall of some aggressive behaviors-1.

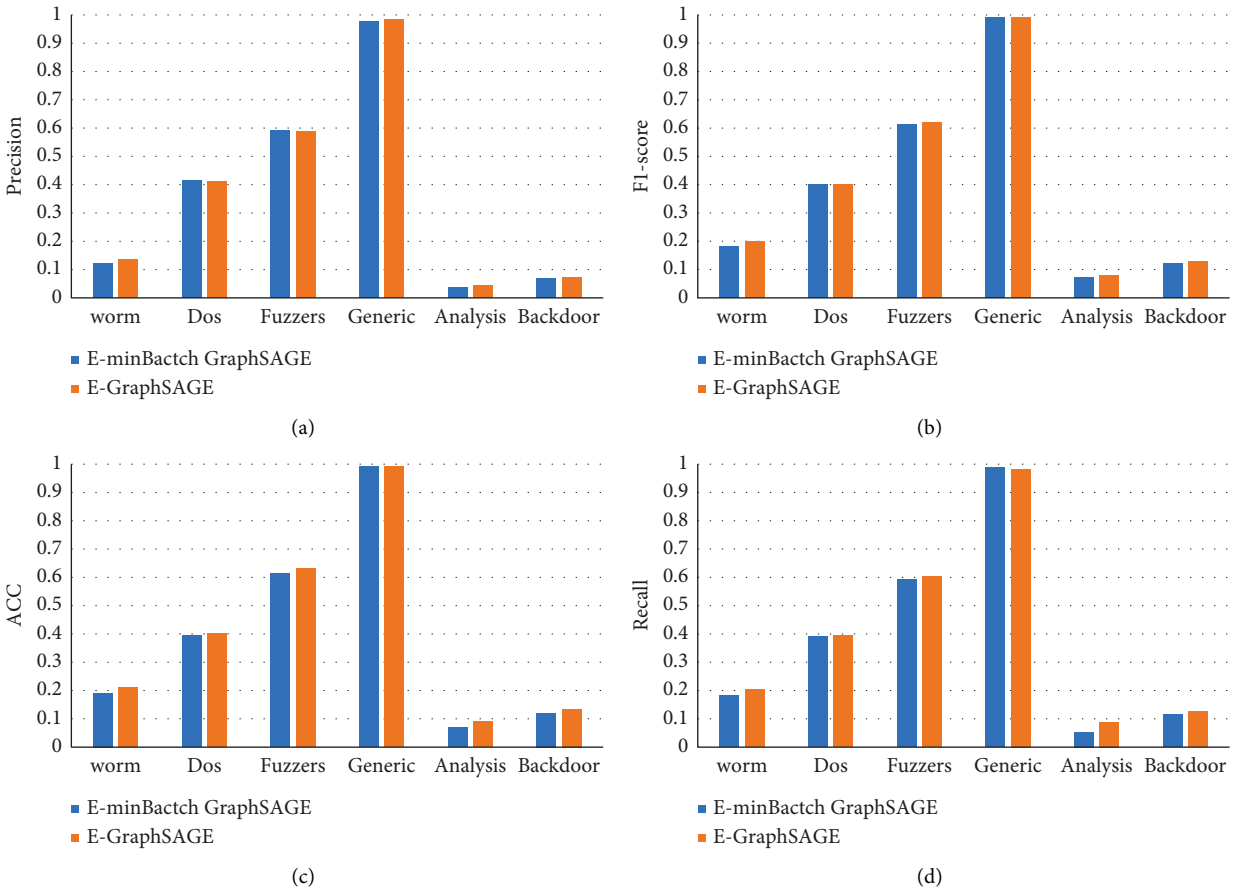


Figure 9: Compare the ACC, *F*1-score, precision, and recall of the model proposed in this paper with E-graph SAGE on three attacks: worm, dos, fuzzers, generic, analysis, backdoor. (a) Precision of some aggressive behaviors-2. (b) *F*1-score of some aggressive behaviors-2. (c) ACC of some aggressive behaviors-2. (d) Recall of some aggressive behaviors-2.

Calculate the confusion matrix to show the effect of the E-minBatch GraphSAGE model. The confusion matrix is shown in Figure 7.

The E-minBatch GraphSAGE model proposed in this paper achieves better results than the E-GraphSAGE model in the detection of three attack behaviors: Shellcode, Reconnaissance, and Exploits. As shown in Figure 8(a), the detection rate of Shellcode attack increased by 2.65%, the detection rate of Reconnaissance attack increased by 1.48%, and the detection rate of Exploits attack increased by 2.83%. At the same time, as shown in Figure 8, the model proposed in this paper still has a certain degree of improvement compared to the E-GraphSAGE model in other metrics (ACC, *F*1-score, and Recall).

To make the model better adapt to the complex network environment, when training the E-minBatch GraphSAGE model, a presampling process is performed, resulting in when the remaining attack behaviors of the UNSW-NB15 dataset are used, and the effect obtained by the model proposed in this paper is similar to that obtained by the E-GraphSAGE model, as shown in Figure 9.

6. Conclusion

This paper proposes a new algorithm-E-minBatch GraphSAGE based on E-GraphSAGE. To make the model better adapt to the complex network environment, the E-minBatch GraphSAGE algorithm presamples the neighbor edges of each node of the model after the graph structure data is constructed. In order to verify the effect of E-minBatch GraphSAGE, experiments are carried out on the UNSW-NB15 dataset. The results show that the algorithm proposed in this paper is comparable to the E-GraphSAGE algorithm in terms of attack behavior detection accuracy and *F*1-score in a complex network environment. In comparison, the model's accuracy and *F*1-score have achieved better results. Compared with the current state-of-the-art deep learning algorithms, the algorithm proposed in this paper is still insufficient in terms of accuracy. At the same time, the algorithm proposed in this paper has great problems in small sample detection, which are worthy of further study.

Data Availability

The data set can be accessed from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by National Natural Science Foundation of China (Grant no. 62102049), "Research on Intelligent Depth Detection of APT Attacks for Cyber-Physical Systems," the National Natural Science Foundation of China (no. 62076042), the Key Research and Development Project of Sichuan Province (nos. 2021YFSY0012, 2020YFG0307, and 2021YFG0332), the Science and Technology Innovation Project of Sichuan (no. 2020017), the Key Research and Development Project of Chengdu (no. 2019-YF05-02028-GX), the Innovation Team of Quantum Security Communication of Sichuan Province (no. 17TD0009), and the Academic and Technical Leaders Training Funding Support Projects of Sichuan Province (no. 2016120080102643).

References

[1] K. W. Schmidt and Schmidt, "Distributed real-time protocols for industrial control systems: framework and examples," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 10, pp. 1856–1866, 2012.

[2] J. E. Rubio, R. Roman, and J. Lopez, "Integration of a threat traceability solution in the industrial Internet of things," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 10, pp. 6575–6583, 2020.

[3] Y. Hu, A. Yang, H. Li, Y. Sun, and L. Sun, "A survey of intrusion detection on industrial control systems," *International Journal of Distributed Sensor Networks*, vol. 14, no. 8, p. 155014771879461, August 2018.

[4] R. Langner, "Stuxnet: dissecting a cyberwarfare weapon," *IEEE Secur Priv2011*, vol. 9, no. 3, pp. 49–51, 2021.

[5] R. M. Lee, M. J. Assante, and T. Conway, *Analysis of the Cyber Attack on the Ukrainian Power Grid*, p. 2, Electricity Information Sharing and Analysis Center(E-ISAC), Washington,DC, 2020.

[6] J. Staggs, *Adventures in Attacking Wind Farm Control Networks*, black hat, San Francisco, CA, 2017.

[7] E. Noonan, *Colonial Pipeline Didn't Have Multifactor Authentication in Place—And Most Defense Contractors Don't Either*Nextgov.com, China, 2021.

[8] R. Singh, H. Kumar, R. K. Singla, and R. R. Ketti, "Internet attacks and intrusion detection system," *Online Information Review*, vol. 41, no. 2, pp. 171–184, 2017.

[9] W. Lee and S. J. Stolfo, "Data mining approaches for intrusion detection," in *Proceedings of the 7th USENIX Security Symposium*, pp. 120–132, San Antonio,TX, January 1998.

[10] L. Khan, M. Awad, and B. M. Thuraisingham, "A new intrusion detection system using support vector machines and hierarchical clustering," *The VLDB Journal*, vol. 16, no. 4, pp. 507–521, 2007.

[11] E. Hodo, X. Bellekens, A. Hamilton et al., "Threat analysis of iot networks using artificial neural network intrusion detection system," in *Proceedings of the 2016 International Symposium on Networks Computer and Communications*, pp. 1–6, China, May 2016.

[12] A. Diro and N. Chilamkurti, "Distributed attack detection scheme using deep learning approach for Internet of Things," *Future Generation Computer Systems*, vol. 282, pp. 761–768, May 2017.

[13] R. Beghdad, "Critical study of neural networks in detecting intrusions," *Computers & Security*, vol. 27, no. 5-6, pp. 168–175, 2008.

[14] W. W. Lo, S. Layeghy, M. Sarhan, and E. GraphSAGE, "A Graph Neural Network Based Intrusion Detection System," 2021, https://arxiv.org/abs/2103.16329.

[15] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, vol. 02216, 2017.

[16] A. Khraisat, I. Gondal, P. Vamplew, J. Kamruzzaman, and A. Alazab, "A novel ensemble of hybrid intrusion detection system for detecting Internet of things attacks," *Electronics*, vol. 8, no. 11, p. 1210, 2019.

[17] X. Li, M. Xu, P. Vijayakumar, N. Kumar, and X. Liu, "Detection of LowFrequency and Multi-Stage Attacks in Industrial Internet of Things," *IEEE Transactions on Vehicular Technology*, vol. 69, 2020.

[18] A. A. Süzen, "Developing a multi-level intrusion detection system using hybrid-DBN[J]," *Journal of Ambient Intelligence and Humanized Computing*, 2020.

[19] W. Liang, K. C. Li, J. Long, X. Kui, and A. Y. Zomaya, "An industrial network intrusion detection algorithm based on multifeature data clustering optimization model," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 3, pp. 2063–2071, 2020.

[20] D. Huang, X. Shi, and W. A. Zhang, "False data injection attack detection for industrial control systems based on both time and frequency-domain analysis of sensor data[J]," *IEEE Internet of Things Journal*, no. 99, p. 1, 2020.

[21] K. Zhang, C. Shen, H. Wang, Z. Li, Q. Gao, and X. Chen, "Cluster computing data mining based on massive intrusion interference constraints in hybrid networks[J]," *Cluster Computing*, vol. 22, no. 3, pp. 7481–7489, 2019.

[22] A. Alharbi, W. Alosaimi, H. Alyami, H. T. Rauf, and R. Damaševičius, "Botnet attack detection using local global best Bat algorithm for industrial Internet of things," *Electronics*, vol. 10, no. 11, p. 1341, 2021.

[23] M. H. Ali, M. M. Jaber, S. K. Abd et al., "Threat analysis and distributed denial of service (DDoS) attack recognition in the Internet of things (IoT)," *Electronics*, vol. 11, no. 3, p. 494, 2022.

[24] M. Wozniak, J. Silka, M. Wieczorek, and M. Alrashoud, "Recurrent Neural Network model for IoT and networking malware threads detection[J]," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 99, 2020.

[25] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[26] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," *Advances in Neural Information Processing Systems 30 (NIPS 2017*, China, 2017.

[27] K. Xie, Y. Yang, Y. Xin, and G. Xia, "Cellular neural network-based methods for distributed network intrusion detection," *Mathematical Problems in Engineering*, vol. 2015, no. 3, pp. 1–10, Article ID 343050, 2015.

[28] Y. Y. Huang, D. Wang, Y. Sun, and B. Hang, "A fastin tra-coding algorithm for HEVC by join tly utilizing naïve Bayesian and SVM," *Multimedia Tools and Applications*, vol. 79, no. 45, pp. pp33957–33971, 2020.

[29] N. Moustafa and J. Slay, "UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set)," *Proc. Mil. Commun. Inf. Syst. Conf. (MilCIS), Nov.,* vol. 12, pp. 1–6, 2015.

[30] J. Toldinas, A. Venčkauskas, R. Damaševičius, Š. Grigaliūnas, and Morkevičius, "A novel approach for network intrusion detection using multistage deep learning image recognition," *Electronics*, vol. 10, no. 15, p. 1854, 2021.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 7585457, 10 pages
https://doi.org/10.1155/2022/7585457

Research Article

Light Weighted CNN Model to Detect DDoS Attack over Distributed Scenario

Harish Kumar,[1] **Yassine Aoudni**,[2] **Geovanny Genaro Reivan Ortiz**,[3] **Latika Jindal**,[4] **Shahajan Miah**,[5] **and Rohit Tripathi**[6]

[1]*Department of Computer Science, College of Computer Science, King Khalid University, Abha 61413, Saudi Arabia*
[2]*Department of Computers and Information Technology, College of Sciences and Arts in Turaif, Northern Border University, Arar, Saudi Arabia*
[3]*Laboratory of Basic Psychology, Behavioral Analysis and Programmatic Development PAD-LAB, Catholic University of Cuenca, Cuenca, Ecuador*
[4]*Department Computer Science Engineering, Medi-Caps University, Indore, India*
[5]*Department of EEE, Bangladesh University of Business and Technology (BUBT), Dhaka, Bangladesh*
[6]*Electronics Engineering Department, J C Bose University of Science and Technology YMCA, Faridabad, India*

Correspondence should be addressed to Shahajan Miah; miahbubt@bubt.edu.bd

Received 28 March 2022; Revised 8 May 2022; Accepted 17 May 2022; Published 13 June 2022

Academic Editor: Robertas Damaševičius

Copyright © 2022 Harish Kumar et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

The minimal-degree distributed denial-of-service attack takes advantage of flaws in the adaptive mechanisms of network protocols, which could have a big impact on network service quality. It is very hard to find, has a low attack rate, and comes at a set time. Detection methods that have been used before have problems because they only use one type of detection and are not very good at identifying the object. In the end, a way to detect many sorts of minimal DDoS assaults that use deep hybrid learning is suggested. To construct multi-type limited DDoS threat data sets and mimic diverse sorts DDoS assaults and legitimate traffic in varying situations in the 5G setting, collect congestion at the networking entry and extract flow feature info are considered. From a statistical threshold and feature engineering point of view, these data sets show how many sorts of minimal DDoS assaults are there. This study aims to develop a deep hybrid learning-based multi-type low-rate DDoS attack detection solution for 5G networks which is the novel model that is recently deployed, and a hybrid deep learning algorithm was used to train the algorithm offline, and the algorithm's performance was compared to that of the LSTM-Light GBM and LSTM-RF algorithms. The CNN-RF revealing model was then used to detect minimal DDoS assaults at the gateway, so that multiple attacks could be detected at the same time. It can identify 4 sorts of low-rate DDoS assaults like Slow-Headers, Slow-Body, Slow-Read, and Shrew assaults, in a 120-second window. The false intercept rate is 11.03 percent. This means that 96.22 percent of traffic could be found. Using the strategy suggested can help cut down on the traffic concentration of minimal DDoS attacks at the net ingress. It can also be used in real-world situations.

1. Introduction

The DDoS assault is the large-scale distributed and very damaging network attack approach that may adversely damage service availability. It has progressively grown among the utmost severe security risks to the web. With the continual innovation and updating of attack technology, a new assault variation, called a low-rate DDoS attack, is developed. This attack makes use of flaws in the network protocol adaptive mechanism to deliver attack packets at a lower rate, lowering the victim's service quality. It has good concealment and a low attack rate. There are low-rate/minimal DDoS assaults of many protocols in the network environment as well as periodic and aperiodic attack methods [1]. As a result, effectively identifying many forms of minimal DDoS assault traffic is an important challenge that must be addressed.

This research primarily offers a multi-type low-rate DDoS assault revealing approach for networks in the 5G context based on deep hybrid learning. First, experimental

data sets are obtained by simulating various sorts of low-rate assaults and normal communication behaviour; then, the characteristic information of various types of low-rate DDoS assaults is analyzed, and feature selection is performed based on the usual information; finally, the detection model is realized by combining the hybrid deep learning algorithm. Finally, the detection model is placed at the network's entry to enable online detection of many sorts of low-rate DDoS assaults.

This study's key contributions are as follows:

(1) Various forms of low-rate DDoS assaults and normal communication in diverse settings are simulated in the 5G environment, network traffic characteristic information during a specific time is gathered, and a tagged minimal-degree DDoS assault data set is generated.

(2) A multi-type low-rate DDoS assault feature set is suggested. The characteristic information of several forms of low-rate DDoS assaults and ordinary traffic is investigated from the standpoint of statistical thresholds and feature engineering, and 40 effective minimal-degree DDoS assault characteristics are derived.

(3) A multi-type low-rate DDoS assault detection approach is provided. The offline training, deployment, and detection of hybrid deep learning models are implemented using the low-rate DDoS assault feature set. The detection findings demonstrate that by choosing the ideal time frame, the approach presented in this study can efficiently identify four forms of minimal DDoS assaults, namely, Slow-Headers attack, Slow-Body attack, Slow-Read attack, and Shrew attack.

2. Related Work

For a long time, the research on minimal-degree DDoS assaults has received extensive attention from scholars at home and abroad. At the beginning of the 21st century, Kuzmanovic proposed the definition of Shrew attack, collected relevant data of minimal-degree DDoS assaults, and conducted appropriate analysis and research [2]. The research on minimal-degree DDoS assault revealing and defense mainly includes twofold methods. One is the detection method based on statistical analysis. The authors proposed a minimal-degree DoS assault revealing method centred on the Pearson relationship, which uses the Pearson coefficient of correlation based on the Hilbert spectrum net congestion, to characterize network traffic information, and compares this information with a threshold to detect low-rate attacks against TCP [3]. Author analyzed the sequence similarity between the minimal-degree DDoS assault pulses at the victim end from the perspective of sequence matching, used the Smith–Waterman algorithm, and designed a double-threshold rule to detect TCP-based low-rate attacks [4]. The authors proposed a method based on network self-similarity to analyze the impact of low-rate attacks on traffic self-similarity and used H-index combined with thresholds to identify attacks and legitimate traffic [5]. The deep neural model (DNN) is proposed as a deep learning technique for malware detection on a subset of frames acquired from data transfer [6]. The method suggested by the researchers limits the cost of interference in IoT transmitting data, and the network's smart use of training sets efficiently differentiates the conventional and threat sequences [7]. The above methods for detecting low-rate attacks only see low-rate attacks based on TCP and depend on the set of points, which are easily affected by the randomness of the network environment and cannot achieve excellent detection results.

Another kind is machine learning-based detection, which uses traffic properties and M-L procedures to identify minimum degree DDoS attacks. The authors recommended an approach on the fundamentals of principal factor investigation and S-V-M to sense minimal-degree TCP assaults. The major component analysis tactic effectively captures network communication properties while filtering noise from the environment [8]. The authors proposed a minimal-degree DDoS assault detection method for TCP in edge environments, which used local complex feature mining and deep CNN to acquire the finest trait distribution of raw info automatically, and deep reinforcement learning Q networks as decision-making to improve attack detection decision-making accuracy [9]. The authors constructed a minimal-degree DDoS assault detection system based on decomposition machines, offered a feature combination mechanism, established the correlation between feature samples, and detected HTTP-based low-rate assaults. J48, random tree, REP tree, random forest, multilayer perceptron, and support vector machine are six models that detect HTTP-based minimal-degree DDoS assaults, according to Reference, which proposes using machine learning approaches to identify low-rate DDoS assaults in the SDN situation [10]. DNN models can perform efficiently and precisely although with small samples since its architecture includes segmentation method and identification procedures, and also strands that upgrade themselves as they are programmed [6]. This method, however, has a higher false-positive rate than DDoS assaults. Hybrid deep learning algorithms may fully use the advantages of machine learning and deep learning algorithms. This article includes multiple machine learning models to anticipate application layer DDoS assaults in real time [11]. The authors have proposed CyDDoS architecture for an automated intrusion detection system (IDS) that blends a feature map synthesis algorithm with such a neural network [12].

A hybrid based on a long-short-term-memory network and a CNN was suggested by researcher. Therefore, successfully implementing security strategy to prevent a system from this danger is a significant issue since DDoS employs a variety of attack methods with numerous conceivable combinations [13]. The deep learning architecture detects Bot, Post Scan, and XSS threats in the CICIDS2017 data set. The detection system has been proved to have better detection capabilities [14]. The authors proposed a deep learning-based hybrid anomaly detection system that uses the limited Boltzmann machine and support vector machine methods to reduce the data's feature dimensions, but the

data set used in the investigation was KDD99, which is incorrect. At a finer level, DoS assaults are categorized and identified. The authors proposed a hybrid time-series forecasting model for stock forecasting based on an extended short-term memory network and LightGBM, which performed well [15]. In terms of prediction, author proposes a hybrid deep learning model based on an extended short-term memory network and random forest (RF, random forest), which outperforms a single machine learning strategy [16]. Minimal-degree DDoS assault revealing approaches, such as the ones given above, can only identify a single sort of minimal DDoS assaults, which has the drawbacks of only detecting one type of attack and low detection accuracy. Given the aforementioned limitations, this research proposes a CNN-RF hybrid deep learning-based minimal-degree DDoS assault revealing system that can learn the characteristics of many kinds of attack traffic and improve the accuracy of online detection of numerous sorts of minimal-degree DDoS assaults.

3. Characteristic Analysis of Minimal-Degree DDoS Assaults

In this study, minimal-degree DDoS assaults are classified into two types: HTTP-based low-rate DDoS attacks and TCP-based minimal-degree DDoS assaults [17].

Slow-Headers, Slow-Body, along with Slow-Read assaults are examples of HTTP-based minimal-degree DDoS assaults [18]. This sort of assault exploits the weakness in the current HTTP Keep-Alive method, maintains the connection for an extended period of time, and continually consumes resources of server, ensuing in a service denial to the Web server. Among these, the Slow-Headers attacker sends an unfinished HTTP request ending with the character "rn," causing the server to believe that the request was not delivered and continuing to wait. Finally, the number of connections approaches the server's maximum capacity, and the new request is unable to be handled, resulting in a rejection-service assault. The sluggish body attacker makes a POST request to the server with a large content-length value. Even yet, the server only delivers a tiny amount of bytes each time, and the server's resources are depleted when requests exceeds an assured threshold. Finally, Slow-Read attackers submit valid requests to the server to read huge data files while setting the TCP sliding window to a low number. As a consequence, establishing a communication link between the server and the attacker takes a lengthy time. When the number of connections exceeds a certain threshold, the service cannot be supplied.

TCP-based low-rate DDoS assaults come in a variety of flavors. This research focuses on the Shrew attack, which leverages the TCP timeout retransmission mechanism to transmit high-speed burst packets on a regular basis, lowering the victim's quality of service and performance. The suggested model overcomes it by incorporating a novel position-oriented neural layer [19]. This article mostly replicates four forms of minimal-degree DDoS assaults using attack tools and Python scripts: Slow-Headers assaults, Slow-Body assaults, Slow-Read assaults, and Shrew assaults.

A typical analysis of minimal-degree DDoS attacks is mostly based on the original minimal-degree DDoS assaults. The CICFlowMeter feature extraction program extracts comprehensive bidirectional flows based on time frames, reflecting properties such as forward and reverse data flows. This technique is used as our research is mainly aimed on the attack tools namely as Slow-Headers attack, Slow-Body attack, Slow-Read attack, and Shrew attack; however, this work mostly replicates four forms of low-rate DDoS assaults. Aside from tag values, the device produces a total of 83 other types of feature information, such as flow ID, quintuple information, stream-level features, and package-level features. The flow ID is a penta-tuple consisting of the birthplace IP address, purpose IP address, port location, destiny port, and procedure that is used to uniquely identify the flow. Stream-level characteristics include statistics regarding the stream's time, duration, and bytes per second. The amount of forwarding/reverse packets per second, statistical factors of packet length, SYN/FIN/RST flag bit count, and so on are all packet-level characteristics.

4. Minimal-Degree DDoS Assault Detection Framework

This section first introduces the composition of the detection framework, then introduces the principle and implementation of the data set generation module, and finally presents the specific performance and critical technologies of the offline training module and online detection module of the hybrid deep learning model detail. The detection framework comprises a data set generation module, feature analysis and selection module, a detached training unit, and a connected detection unit. The minimal-degree DDoS attack detection framework is shown in Figure 1. The framework is divided into data processing and deep hybrid learning. Figure 2 shows the flowchart for the proposed methodology.

The data processing part is responsible for preliminary processing of the acquired network traffic and is divided into a data set generation module and feature analysis and selection module. The data set generation module is used to obtain network traffic in a specified period, extract flow feature information, and perform data cleaning to get minimal-degree DDoS assault data set containing 4 types of minimal-degree DDoS attacks and regular traffic. The trait analysis and selection module analyzes the trait information of different kinds of minimal-degree DDoS assault from statistical thresholds and trait engineering and summarizes the valuable features of multiple types of minimal-degree DDoS assault.

The deep hybrid learning component detects many sorts of minimal-degree DDoS assaults and is separated into twofold segments: disconnected training and connected detection. The disconnected training unit selects valuable features from the data set for feature selection, uses a hybrid deep learning algorithm for training and testing, performs performance evaluation and related parameter optimization based on classification results, and selects the best attack detection model. By recording traffic in real time, the online

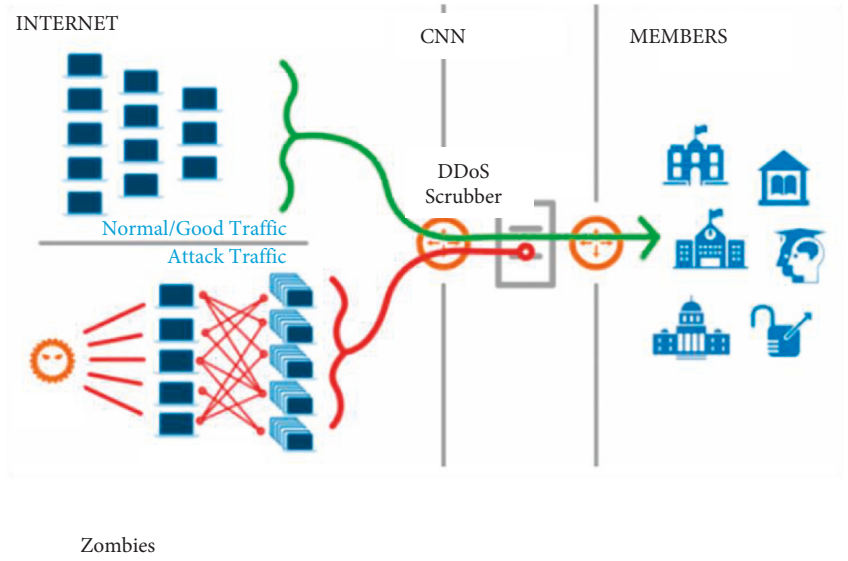


Figure 1: Minimal-degree DDoS assault detection framework.

detection module deploys the trained hybrid deep learning detection model to the network entry and achieves connected revealing of different forms of minimal-degree DDoS assaults. A model's output information is employed to recognize minimal-degree DDoS assaults on traffic to be detected—a particular sort of attack.

4.1. Data Processing Part

4.1.1. Data set Generation Module. The data set generation module is used to obtain the network traffic in a certain period. Then, the flow feature information is extracted by the flow feature extraction tool CICFlowMeter to get a minimal-degree DDoS assault data set. This data set contains multiple sorts of minimal-degree DDoS assaults and regular communication congestion in 5G environ, reflecting the traffic patterns in natural environments.

The generated a hefty figure of regular transmission simulation requests according to the third-generation cooperation project (3GPP) and IEEE for actual traffic laws of devices in different 5G application scenarios [20, 21]. This rule is obtained through the traffic data collected in the real scene. The result includes the influence of various environmental factors, which can reflect the request situation in the exact location. In this study, the method is improved to generate regular communication traffic. Combined with the four minimal-degree DDoS assault traffic generated by outbreak tools as well as scripts, a new minimal-degree DDoS assault data set will be obtained.

As per this study, attack is realized by sending traffic through attack tools. Considering the security of the network environment, the capture of low-rate network traffic is recognized based on the VMware vSphere virtualization experimental platform. The realistic environment is close to the natural environment, reflecting the traffic statistics in the virtual environment. Thereafter, the traffic collection tool Tcpdump is deployed and installed to capture the data

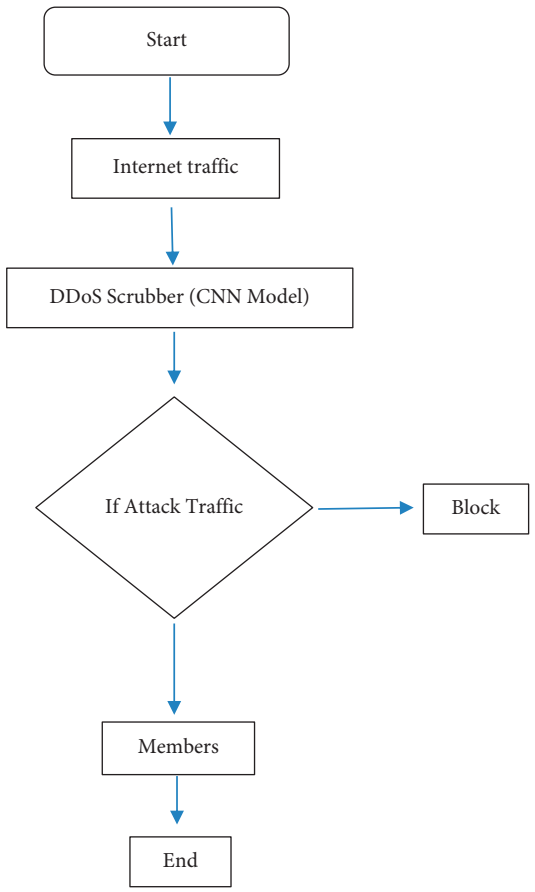


Figure 2: Flowchart.

packets in the network. The data set collection point is at the access gateway of the network entrance, which can completely capture the communication traffic in the network. Finally, CICFlowMeter is used to extract characteristic information of network traffic. At the same time, according to the attack plan in Table 1, the extracted feature information

Table 1: Minimal-degree DDoS attack plan.

Attack time	Source IP	Destination IP	Traffic type
2021.5.25	23.1.0.22	23.1.1.22	Slow-Headers
	23.1.0.12	23.1.1.23	Slow-Body
	23.1.0.13	23.1.1.24	Slow-Read
15 : 35–16 : 15	23.1.0.14	23.1.1.25	Shrew
	23.1.0.20~23.1.0.29	23.1.1.73	Normal flow

is labeled, and the labeled data set is used for the training and verification of the detection model. This article includes multiple machine learning models to anticipate application layer DDoS assaults in real time. The authors have proposed CyDDoS, an architecture for an automated intrusion detection system (IDS) that blends a feature map synthesis algorithm with such a neural network [22]. The three types of minimal-degree DDoS attack methods, Slow-Headers assault, Slow-Body assault, and Slow-Read assaults studied in this article, send the attack traffic by modifying the parameters of the slow Http test and slow HTTP attack tool, and the Shrew attack realizes the sending attack by writing Python scripts flow. Python scripts are used for regular communication requests based on the statistical laws of different scenarios in the 5G environment to simulate sending massive connection regular request traffic. Based on the above implementation methods, this study collects traffic and automatically extracts flow feature information under minimal-degree DDoS assault and normal communication behaviour [23]. In our investigation, the capture period starts at 08:00 on May 19, 2021 and ends at 17:00 on May 24, 2021. During this period, different attacks were launched, including low-rate DDoS assaults, DDoS network stratum assaults, DDoS application stratum assaults, and distributed reflection amplification attacks. Table 1 shows the attack plan for minimal-degree DDoS assaults.

Based on the network traffic pcap file obtained by the above attack plan, the traffic feature extraction tool CICFlowMeter is employed to excerpt the traffic trait info, and a multi-type minimal-degree DDoS assault data set is obtained. Table 2 depicts the quantity of data samples of every single traffic type in the data set and the ratio of standard traffic samples. It can be seen that the number of data samples of regular traffic is plentiful superior than the count of data samples of each minimal-degree DDoS attack, reflecting the minimal-degree DDoS attacks.

5. Experiment and Result Analysis

This study simulates various minimal-degree DDoS attacks and regular communication requests in the 5G environment. It conducts performance evaluations of different hybrid deep learning detection models and online detection performance tests under other detection time windows. Table 2 displays the number of data samples from each traffic category in the data set as well as the ratio of regular traffic data. Figure 3with Tables 3 and 4depicts the efficiency and $F1$ value of the three models. As shown in Figure 4, for detecting Slow-Headers attack traffic, the CNN-RF model outperforms the other two models in terms of effectiveness and F1 value for identifying ordinary benign traffic.

Table 2: Number and proportion of data samples for each traffic type.

Traffic type	Number of data samples	The proportion of attack traffic to normal traffic
Slow-Headers	100 793	01 : 04.5
Slow-Body	110 044	01 : 04.5
Slow-Read	68 074	01 : 04.5
Shrew	45 389	01 : 04.5
Normal flow	460 619	—

5.1. Experimental Environment. To authenticate the revealing effect of the technique in this research on multi-type low-rate DDoS attacks, a related test platform is built on the network platform using actual network equipment.

In this study, a virtual platform based on Vmware vSphere is set up as the experimental environment. A total of nine hosts were used in the experiment, including two routers, one client host, four dummy hosts, and two web servers. The investigation in this study builds a hybrid deep learning model based on the TensorFlow framework. The programming language is Python3.8, and the machine learning library of TensorFlow2.1 and Keras2.2.4 is used to build the model. The Ubuntu18.04 is software background in server operating structure, and the number of virtual cores is 8, the memory is 8 GB, four hosts are used as puppet hosts, and two virtual machines built with web servers are used as attacked servers. This is critical to halt fraudulent activity since they have a long-term influence on financial circumstances. Outlier detection has several essential applications for fraud prevention [24]. Detection is performed at the network entry router, and data collection and cleaning functions are provided.

The simulation includes public services, smart homes, PC Internet access, and MTC communication based on this connection.

The four transmission scenarios generated a large number of regular communication data requests. Minimal-degree DDoS assault attacker controls four puppet hosts to periodically send minimal-degree DDoS attacks based on HTTP protocol and TCP protocol to the web server. The experimental minimal-degree DDoS assault types select HTTP-based Slow-Headers assaults, Slow-Body assaults, Slow-Read assaults, and TCP-based Shrew assaults [25].

5.2. Evaluation Indicators. The minimal-degree DDoS assault detection framework implements offline training and online detection for various kinds of minimal-degree DDoS assault data based on hybrid learning procedure [26]. Offline activity mainly analyzes the model's classification performance through six evaluation indicators: accuracy, precision, recall, $F1$ value, detection time, and confusion matrix. Among them, the rate of exactness symbolizes the ratio of

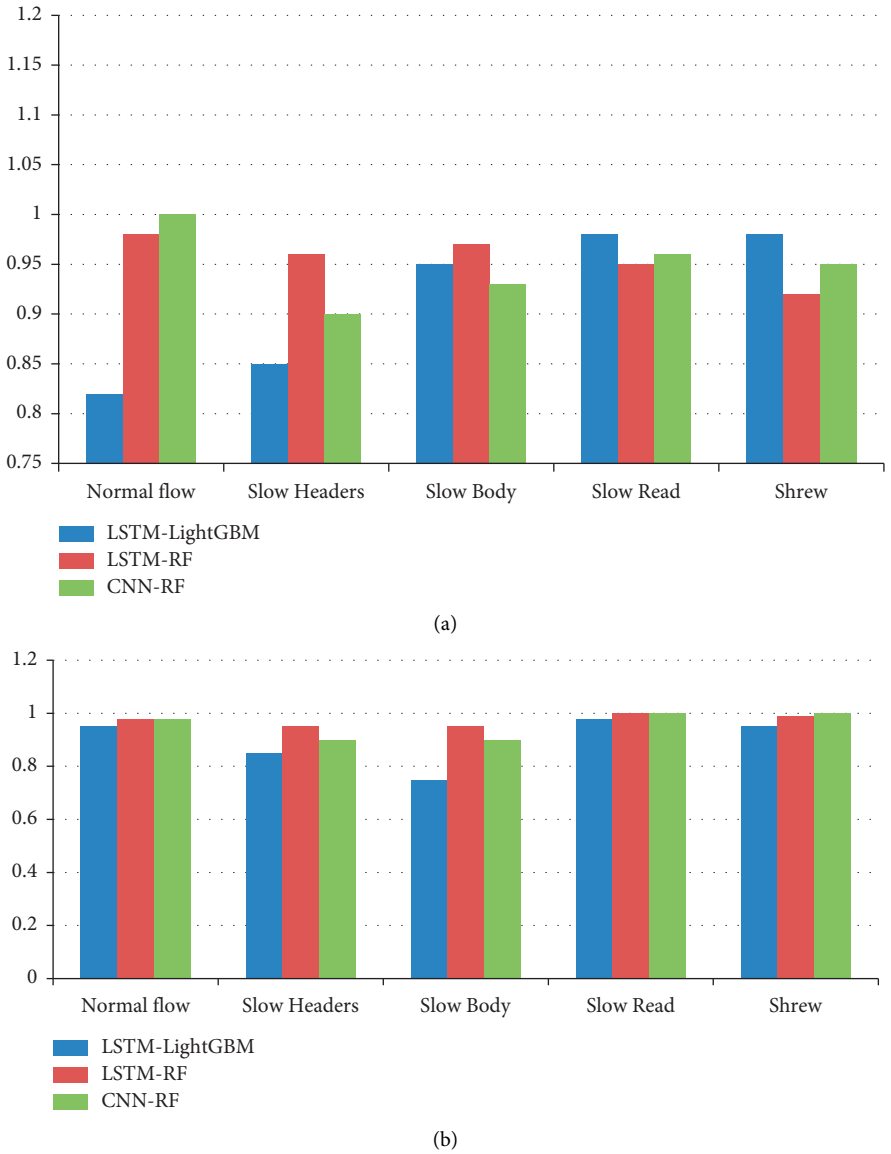


Figure 3: Comparison of precision and $F1$ scores of different models. (a) Comparison of the accuracy of different models. (b) Comparison between $F1$ of different models.

Table 3: Comparison of the accuracy of different models.

$F1$ value	LSTM-LightGBM	LSTM-RF	CNN-RF
Normal flow	0.82	0.98	1
Slow-Headers	0.85	0.96	0.9
Slow-Body	0.95	0.97	0.93
Slow-Read	0.98	0.95	0.96
Shrew	0.98	0.92	0.95

Table 4: Comparison between $F1$ of different models.

$F1$ value	LSTM-LightGBM	LSTM-RF	CNN-RF
Normal flow	0.95	0.98	0.98
Slow-Headers	0.85	0.95	0.9
Slow-Body	0.75	0.95	0.9
Slow-Read	0.98	1	1
Shrew	0.95	0.99	1

the amount of exact samples classified through the prototype to the overall quantity of pieces; the exactness degree represents the proportion for an amount of samples suggested by prototype as an attack category and the count of samples that are assault kinds; and the recall rate represents the prototype suggested as an attack category [27]. The share of the sum of pieces to all the examples of this assault type are as follows: the $F1$ value combines the results of precision and recall, representing the harmonic average of the two, which can more accurately reflect model performance; detection time reflects the time complexity of the model. It is used to measure the time efficiency of the model; the classification effect of the prototype is examined by employing confusion matrix as well as the grade to which the predicted label matches the actual label, which corresponds to the recall rate numerically [28].

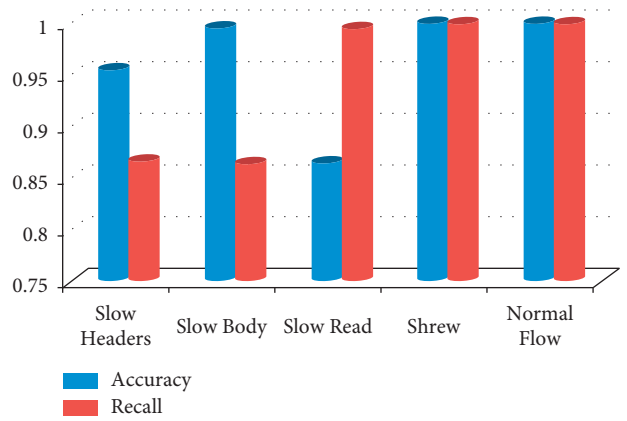


Figure 4: Detection performance.

In addition, to analyze the classification of online detection, new evaluation indicators are defined: false intervention degree and malicious congestion revealing degree used to evaluate an online detection of normal and malicious traffic, respectively. Among them, the false interception rate represents the proportion of misjudging regular traffic as diverse kinds of minimal-degree DDoS assaults, and the calculation is shown in formula (1); the malicious traffic detection rate represents the proportion of detected malicious traffic to the overall count of negative traffic samples, and the calculation is shown in formula (2).

$$\text{false}_{\text{interception}_{\text{rate}}} = \sum_{i=1}^{4} \frac{G_i}{M}, \tag{1}$$

$$\text{Malicious}_{\text{trafficdetection}_{\text{rate}}} = 1 - \sum_{i=1}^{4} \frac{T_i}{\sum_{i=1}^{4} B_i}, \tag{2}$$

where G_i represents the number of data samples that misjudge the regular traffic in the network environment as a further four forms of minimal-degree DDoS assault traffic after online detection; M represents the total number of data samples of regular traffic in the network environment; T_i represents the number of undetected data samples of minimal-degree DDoS assault congestion within the network environment after detection; B_i represents the total number of data samples of different types of minimal-rate DDoS assault congestion within the network environment.

5.3. Offline Training Analysis. Based on the minimal-degree DDoS assault data set obtained by the data set generation module in Section 3, data cleaning is performed, including processing the feature data with null feature values and processing feature data with infinite feature values. Feature selection is carried out according to the 40 useful features shown in Figure 3 and is distributed in a dual sets as training as well as test in a ratio of 7 : 3. The data set is shown in Table 5. The total number of data samples in the minimal-degree DDoS assault data set is 794,919, including 556,444 in the preparation set as well as 238,475 in the training set.

Table 5: Minimal-degree DDoS assault data set.

Data set type	Normal flow samples	Number of attack traffic samples
Training set	288555	267800
Test set	129832	108943

The CNN-RF model showed optimal performance through hyperparameter search, given the same minimal-degree of DDoS assault data set and eigenvalues. At the same time, the CNN-RF prototype projected in this study is associated with the LSTM-LightGBM prototype and the LSTM-RF prototype, and the optimum hybrid deep learning prototype is nominated to identify the connected revealing of multi-type minimal-rate DDoS assaults. This study uses four evaluation indicators: detection time, precision rate, $F1$ value, and confusion matrix. Figure 3 shows the confusion matrix performance of the three hybrid deep learning models. It may be perceived that the recognition precision of LSTM-Light-GBM model for each traffic type varies greatly, especially the recognition accuracy of the Slow-Body attack is only 0.5565, and the false-positive rate of the Slow-Headers attack is 0.2695. The recognition accuracy of the LSTM-RF model for the five types of traffic is better than that of the LSTM-LightGBM prototype, especially the recognition accuracy of the Slow-Read attack is about 0.9992, but it will produce a false-positive rate of 0.0788 when identifying the Slow-Body attack. The accuracy of the CNN-RF model overperforms the LSTM-RF, especially the recognition accuracy of Slow-Read assaults and Shrew attack can reach 0.9999. The recognition accuracy of Slow-Headers attack traffic can also get 0.9566.

Figure 3 with Tables 3 and 4 shows the evaluation of the three prototypes in terms of exactness and $F1$ value. As can be seen from Figure 3, for the identification of regular benign traffic, the CNN-RF prototype outperforms the other two designs in terms of accuracy and $F1$ value; for the detection of Slow-Headers attack traffic, the accuracy of the CNN-RF design is the best. Excellent: the LSTM-RF and LSTM-LightGBM models have similar performance in $F1$ value; for detecting Slow-Body and Slow-Read assault congestion in net, the LSTM-LightGBM design has poor performance in both accuracy and $F1$ score, and the CNN-RF model's performance is poor. Best performing: for Shrew, the detection of attack traffic in the three models is in the two evaluation indicators of good performance.

The detection time comparison of different hybrid deep learning ideas is presented in Table 5. It may be seen from Table 6 that the detection time of the CNN-RF model is 268.3689 s, which is about 9 s longer than that of the LSTM-LightGBM design, and about 40 s more minor than that of the LSTM-RF model. However, the LSTM-LightGBM design is significantly lower than the CNN-RF design in detection accuracy and $F1$ score. Therefore, while the detection time is shorter, the CNN-RF design has better accuracy and $F1$ value for various forms of minimal-rate DDoS assaults and regular congestion.

Combining the above evaluation indicators, it can be concluded that the distinction of LSTM-LightGBM model

Table 6: Comparison of detection time of different models.

Model category	LSTM-LightGBM	LSTM-RF	CNN-RF
Detection time/s	259.8986	308.5964	268.3689

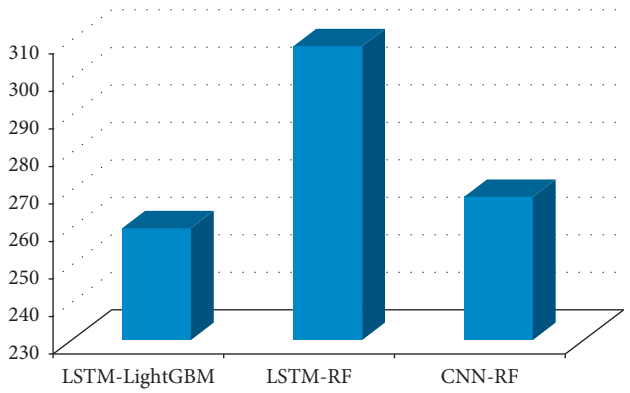


Figure 5: Comparison of detection time.

along with the LSTM-RF model, the CNN-RF model proposed in this article has better performance in regular traffic. Slow-Headers assault, Slow-Body assault, Slow-Read assault and Shrew's assault traffic detection as well as classification all show excellent performance and can accurately detect different types of low-rate DDoS attacks.

5.4. Online Inspection and Verification. The offline training experiments and analysis in Section 4 show that the CNN-RF model has excellent detection performance. To further illustrate that the model version is still the best in online detection, this section compares the performance of LSTM-LightGBM, LSTM-RF, and CNN-RF models in expressions of precision, error interception degree, and malicious traffic detection degree. The contrast of detection time is shown in Figure 5.

Finally, the best-performing and trained model under the optimal time window is selected, and the fine-grained online detection of multi-type low-rate DDoS attacks is deployed. First, multiple types of minimal-degree DDoS assault traffic files online are replayed, Tcpdump is used to internment network congestion inside the specified detection time window, and flow feature information is extracted through CICFlowMeter; then, the structure and parameters of the trained detection model are read and implemented for online detection. The model outputs detection classification labels, actual labels, and malicious traffic IP addresses; finally, based on statistical methods, the model's detection accuracy rate and negative traffic detection rate and other indicators are viewed.

This section compares the performance of the benchmark detection time window of 60 s with the detection time window of 120 s and 180 s and compares the LSTM-LightGBM, LSTM-RF, and CNN-RF models, respectively, and selects the optimal detection model. The optimal detection time window below is the final online detection parameter. Table 7 shows the performance comparison of

Table 7: Comparison of online detection performance of different models under different time windows.

Model name	Time window/s	Accuracy	False interception rate	Malicious traffic detection rate
LSTM-LightGBM	70	0.85394	0.28498	0.85244
	110	0.89384	0.20312	0.87258
	190	0.87698	0.21015	0.88593
LSTM-RF	70	0.90894	0.20591	0.96852
	110	0.92438	0.17419	0.95478
	190	0.89394	0.19875	0.92574
CNN-RF	70	0.9569	0.17058	0.87244
	110	0.95347	0.11789	0.88574
	190	0.98397	0.20591	0.98657

the accuracy, false interception rate, and malicious traffic detection rate of different models under different time windows.

It can be seen from Table 7 that under the time window of 120 s, the LSTM-LightGBM, LSTM-RF, and CNN-RF models all show relatively optimal detection performance. The accuracy of the LSTM-RF model reaches 0.9243, and the malicious traffic detection rate is 0.9193. When the detection time window is 180 s, the accuracy of the LSTM-RF model drops to 0.897 6; simultaneously, the false interception rate increases to 0.192 7, indicating that a huge quantity of regular, benign transportation is misjudged as malicious traffic. Under the time window of 120 s, the LSTM-LightGBM model performed the worst, with an accuracy of only 0.896 5 and a false intercept rate of 0.203 1. For the CNN-RF model, when the online detection time window is 120 s, the minimum false intercept rate is 0.110 3. That is, the proportion of regular traffic being misjudged as malicious traffic is the lowest; at the same time, the negative traffic data samples detected by this detection mechanism are highest. The ratio of the number is 0.962 2. After analysis, the detection time window of 120 s altogether includes the characteristic information of different sorts of minimal-rate DDoS attacks, reflecting the complete minimal-degree DDoS assault activities, thus effectively distinguishing different kinds of minimal-rate DDoS attacks from regular traffic.

Consequently, the detection time window is set to 120 s, and the CNN-RF model with the best performance is deployed to realize online detection. The detection performance for diverse categories of minimal-rate attacks and regular traffic is obtained through the detection, as shown in Table 8. From Table 8 and Figure 4, it can be seen that the precision rate of the CNN-RF hybrid deep learning model for Slow-Headers assaults, Shrew attack, and regular traffic is above 0.95; and for Slow-Read attack and Slow-Body attack traffic, the precision and recall rate are both above 0.86, resulting in fewer misjudgments between the dual attack categories. In summary, detection exactness of the CNN-RF hybrid deep learning model for every kind of minimal-degree DDoS assaults and regular congestion in traffic reaches 0.965 2, which can accurately detect different types of low-rate DDoS attacks online.

Table 8: Online detection performance under 120-s time window.

Traffic type	Accuracy	Recall
Slow-Headers	0.9546	0.8666
Slow-Body	0.9952	0.8639
Slow-Read	0.8647	0.9948
Shrew	0.9998	0.9994
Normal flow	0.9999	0.9995

It can be seen from the above analysis that the CNN-RF hybrid deep learning model proposed in this article has excellent online detection performance and can realize connected revealing of four kinds of minimal-degree DDoS assaults. At the same time, the accuracy degree of each minimal-degree DDoS assaults is above 0.85, which can prevent the attack from causing more damage to the network; the malicious traffic detection rate reaches 0.962 2, and the detection accuracy rate reaches 0.965 2, which can effectively detect the web online. The malicious traffic in the network reduces the concentration of minimal-degree DDoS assault traffic at the ingress network.

6. Conclusion

Aiming at four types of minimal-degree DDoS assaults, this study obtains minimal-degree DDoS assault data sets, analyzes and obtains 40 effective traits of minimal-degree DDoS assaults, and proposes a variable-kind minimal-degree DDoS based on CNN-RF hybrid learning. The attack detection method and online deployment of this model realize connected revealing of variable types of minimal-degree DDoS assaults. Furthermore, an online detection time window is proposed, and the online detection performance is evaluated using false intervention degree and malicious network congestion revealing rate. Experiments show that the prototype based on CNN-RF hybrid deep learning algorithm can accurately detect different types of minimal-degree DDoS assaults. At the identical interval, the revealing method in this study is highly portable, and the minimal-degree DDoS assault data set is used close to the actual situation, which can be deployed and applied in practical environments when the hybrid deep learning model implements training and detection for multi-type low-rate DDoS attacks. The online detection accuracy in different scenarios decreases related to the attack traffic sending rate and the duty cycle of regular traffic in the detection window. In the future, we will study the optimization model and time window and analyze the relationship between time window and data set and feature selection so that the model can better adapt to the environment and have higher accuracy and detection efficiency.

Data Availability

The data shall be made available on request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This work was supported by the King Khalid University Researchers Supporting Project Number (GRP/326/42).

References

[1] S. Yeom and K. Kim, "Improving performance of collaborative source-side DDoS attack detection," in *Proceedings of the 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 239–242, Daegu, Korea (South), September 2020.

[2] W. Sun, Y. Li, and S. Guan, "An Improved Method of DDoS Attack Detection for Controller of SDN," in *Proceedings of the 2019 IEEE 2nd International Conference on Computer and Communication Engineering Technology (CCET)*, pp. 249–253, Beijing, China, August 2019.

[3] B. Jia and Y. Liang, "Anti-D chain: a lightweight DDoS attack detection scheme based on heterogeneous ensemble learning in blockchain," *China Communications*, vol. 17, no. 9, pp. 11–24, 2020.

[4] J. He, Y. Tan, W. Guo, and M. Xian, "A Small Sample DDoS Attack Detection Method Based on Deep Transfer Learning," in *Proceedings of the 2020 International Conference on Computer Communication and Network Security (CCNS)*, pp. 47–50, Xi'an, China, August 2020.

[5] Z. Liu, Y. He, W. Wang, and B. Zhang, "DDoS attack detection scheme based on entropy and PSO-BP neural network in SDN," *China Communications*, vol. 16, no. 7, pp. 144–155, 2019.

[6] A. E. Cil, K. Yildiz, and A. Buldu, "Detection of DDoS attacks with feed forward based deep neural network model," *Expert Systems with Applications*, vol. 169, Article ID 114520, 2021.

[7] M. H. Ali, M. M. Jaber, S. K. Abd et al., "Threat analysis and distributed denial of service (DDoS) attack recognition in the Internet of things (IoT)," *Electronics*, vol. 11, no. 3, p. 494, 2022.

[8] S. Dong and M. Sarem, "DDoS attack detection method based on improved KNN with the degree of DDoS attack in software-defined networks," *IEEE Access*, vol. 8, pp. 5039–5048, 2020.

[9] Y. Chen, X. Chen, H. Tian, T. Wang, and Y. Cai, "A Blind Detection Method for Tracing the Real Source of DDoS Attack Packets by Cluster Matching," in *Proceedings of the 8th IEEE International Conference on Communication Software and Networks (ICCSN)*, pp. 551–555, Beijing, China, June 2016.

[10] X. Liang and T. Znati, "An empirical study of intelligent approaches to DDoS detection in large scale networks," in *Proceedings of the 2019 International Conference on Computing, Networking and Communications (ICNC)*, pp. 821–827, Honolulu, HI, USA, February 2019.

[11] J.-H. Jun, H. Oh, and S.-H. Kim, "DDoS Flooding Attack Detection through a Step-by-step Investigation," in *Proceedings of the 2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*, pp. 1–5, Perth, WA, Australia, December 2011.

[12] M. J. Awan, U. Farooq, H. M. A. Babar et al., "Real-time DDoS attack detection system using big data approach," *Sustainability*, vol. 13, no. 19, Article ID 10743, 2021.

[13] I. Ortet Lopes, D. Zou, F. A. Ruambo, S. Akbar, and B. Yuan, "Towards effective detection of recent DDoS attacks: a deep learning approach," in *Security and Communication Networks*, W. Li, Ed., vol. 2021, Article ID 5710028, 14 pages, 2021.

[14] V. Popovskyy and V. Skibin, "Entropy Methods for DDoS Attacks Detection in Telecommunication Systems," in *Proceedings of the 2014 First International Scientific-Practical Conference Problems of Infocommunications Science and Technology*, pp. 182–185, Kharkov, Ukraine, October 2014.

[15] D. Erhan and E. Anarim, "Istatistiksel Yöntemler Ile DDoS Saldırı Tespiti DDoS Detection Using Statistical Methods," in *Proceedings of the 2020 28th Signal Processing and Communications Applications Conference (SIU)*, pp. 1–4, Gaziantep, Turkey, October 2020.

[16] L. Wang and Y. Liu, "A DDoS Attack Detection Method Based on Information Entropy and Deep Learning in SDN," in *Proceedings of the 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, pp. 1084–1088, Chongqing, China, June 2020.

[17] A. Sanmorino and S. Yazid, "DDoS Attack detection method and mitigation using pattern of the flow," in *Proceedings of the 2013 International Conference of Information and Communication Technology (ICoICT)*, pp. 12–16, Bandung, Indonesia, March 2013.

[18] L. Luo, J. Wang, and L. Jia, "A CGAN-based DDoS Attack Detection Method in SDN," in *Proceedings of the 2021 International Wireless Communications and Mobile Computing (IWCMC)*, pp. 1030–1034, Harbin City, China, June 2021.

[19] K. Mahajan, U. Garg, and M. Shabaz, "CPIDM: a clustering-based profound iterating deep learning model for HSI segmentation," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 7279260, 12 pages, 2021.

[20] R. Arthi and S. Krishnaveni, "Design and Development of IOT Testbed with DDoS Attack for Cyber Security Research," in *Proceedings of the 2021 3rd International Conference on Signal Processing and Communication (ICPSC)*, pp. 586–590, Coimbatore, India, May 2021.

[21] D. Ushakov, M. Vinichenko, and E. Frolova, "Environmental capital: a reason for interregional differentiation or a factor of economy stimulation (the case of Russia)," *IOP conference series: earth and environmental science*, vol. 272, no. 3, p. 032111, 2019.

[22] Y. Chen, J. Hou, Q. Li, and H. Long, "DDoS attack detection based on random forest," in *Proceedings of the 2020 IEEE International Conference on Progress in Informatics and Computing (PIC)*, pp. 328–334, Shanghai, China, December 2020.

[23] S. Nguyen, J. Choi, and K. Kim, "Suspicious Traffic Detection Based on Edge Gateway Sampling Method," in *Proceedings of the 2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 243–246, Seoul, Korea (South), September 2017.

[24] S. Sanober, I. Alam, S. Pande et al., "An enhanced secure deep learning algorithm for fraud detection in wireless communication," *Wireless Communications and Mobile Computing*, vol. 2021, Article ID 6079582, 14 pages, 2021.

[25] D. Erhan, E. Anarım, and G. K. Kurt, "DDoS attack detection using matching pursuit algorithm," in *Proceedings of the 2016 24th Signal Processing and Communication Application Conference (SIU)*, pp. 1081–1084, Zonguldak, Turkey, May 2016.

[26] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "A novel measure for low-rate and high-rate DDoS attack detection using multivariate data analysis," in *Proceedings of the 2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1-2, Bangalore, India, January 2016.

[27] N. R. Nayak, S. Kumar, D. Gupta, A. Suri, M. Naved, and M. Soni, "Network mining techniques to analyze the risk of the occupational accident via Bayesian network," *International Journal of System Assurance Engineering and Management*, vol. 13, pp. 1–9, 2022.

[28] K. Hong, Y. Kim, H. Choi, and J. Park, "SDN-assisted slow HTTP DDoS attack defense method," *IEEE Communications Letters*, vol. 22, no. 4, pp. 688–691, 2018.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 6418420, 19 pages
https://doi.org/10.1155/2022/6418420

Research Article

Memory-Augmented Insider Threat Detection with Temporal-Spatial Fusion

Dongyang Li,[1,2] **Lin Yang**,[2] **Hongguang Zhang**,[2] **Xiaolei Wang**,[2] **and Linru Ma**[2]

[1]*Command and Control Engineering College, Army Engineering University of PLA, Nanjing 211101, China*
[2]*National Key Laboratory of Science and Technology on Information System Security, Institute of System Engineering, Academy of Military Science PLA, Beijing 100039, China*

Correspondence should be addressed to Dongyang Li; dongyangli_nj@126.com

Received 14 February 2022; Accepted 22 March 2022; Published 26 April 2022

Academic Editor: Robertas Damaševičius

Copyright © 2022 Dongyang Li et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Insider threat detection is important for the smooth operation and security protection of an organizational system. Most existing detection models establish historical baseline by reconstructing single-day and individual user behaviors, and then treat any outlier of the baseline as a threat. However, such methods ignore the temporal and spatial correlations between different activities, which result in an unsatisfying performance. To address such an issue, we propose a novel insider threat detection method, namely, Memory-Augmented Insider Threat Detection (MAITD), in this paper. Such an idea is motivated by the observation that the combination of individual model that focuses on historical baseline and group model that represents peer baseline can effectively identify the low-signal yet long-lasting insider threats, and reduce the possibility of false positives. To illustrate, our MAITD captures the temporal and spatial correlation of user behaviors by constructing compound behavioral matrix and common group model, and combines specific application scenarios to integrate the detection results. Moreover, it introduces the memory-augmented network into autoencoder to enlarge the reconstruction error of abnormal samples, thereby reducing the false negative rate. The experimental results on CERT dataset show that the instance-based and user-based AUCs of MAITD reach up to 87.54% and 94.56%, respectively, which significantly outperform previous works.

1. Introduction

With the frequent occurrence of data breaches and espionage incidents, insider threat has become one of the major challenges for system security. According to a recent survey, the number of security incidents caused by insiders has increased by 47% since 2018, and keeps increasing with increased economic uncertainty [1]. However, with so much at stake, only 33% organizations believe they are capable of detecting abnormal behaviors within the system [2]. Meanwhile, since the initiators of insider threats are typically authorized employees who clearly know the system framework and security measures, such insider damages are more harmful than external attacks. The Cybersecurity Insider organization even declared that "today's most damaging security threats do not originate from malicious outsiders or malware but from trusted insiders with access to sensitive data and systems—both malicious insiders and negligent insiders" [3]. Therefore, in the face of severe practical challenges, it is urgent to propose effective insider threat detection models to prevent such threats.

According to the latest definition given by the CERT Coordination Center, insider threats refer to threats that are carried out by malicious or unintentional insiders, whose authorized access to the organization's network, system, and data is exploited to negatively affect the confidentiality, integrity, availability, and physical well-being of the organization's information, information systems, and workforce [4]. The insiders generally consist of malicious traitors, hypocritical masqueraders, and unintentional perpetrators. Their attack methods include system damage, data breaches, intellectual property theft, etc. Although the topic of insider

threat detection has been studied for long, locating malicious behaviors precisely is still nontrivial and remains an open challenge.

Since threat scenarios are widely varying, it is impractical to explicitly model malicious threats. Consequently, most existing methods tend to convert the insider threat detection into user behavior anomaly detection problem [5]. To illustrate, security analysts build normal user behavior model by analyzing historical data, and regard the outliers (data out of distribution) as threats. Such methods are based on the assumption that adversarial activities do not follow past habitual patterns, which is followed by us in this paper. Many classical unsupervised learning algorithms have been used to model normal user behavior. Among them, autoencoder quickly became the mainstream insider threat detection algorithm because of their robustness on domain knowledge and strong anti-interference ability [6].

In fact, the performance of insider threat detection depends on not only the anomaly detection algorithm but also the representation quality of user behavior. In our previous work [7], we performed related studies on the feature extractions of user behavior, and categorized them into two types: (i) statistical features based on artificial definition; (ii) hidden features based on representation learning. Although previous methods [8–13] have their own unique insights, they are still faced with the following limitations:

> Existing methods ignore the temporal statistics when representing user behavior, thereby limiting the performance. Although some works [8–10] attempted to address this issue, they only capture the temporal characteristics from a single perspective, i.e., the potential sequence relationship and the variation tendency of activity frequency.
>
> Most baseline models do not consider the spatial correlation from their peers' data, which leads to the collective behavior changes caused by occasional factors such as service outages or environmental changes being misidentified as anomalies, thus increasing unnecessary manual investigation costs.
>
> Previous detection models focus on how to reasonably represent user behaviors, while ignoring the impact of optimizations on anomaly detection algorithms. Such a one-sided preference leaves much room for improvement.

To address the above limitations, we propose a novel insider threat detection model named Memory-Augmented Insider Threat Detection (MAITD). Our model first adopts the frequencies of daily activities as basic features, then employs the temporal-spatial fusion and an unsupervised learning algorithm to improve the overall performance. The so-called temporal-spatial fusion aims at capturing the temporal and spatial statistics of user behaviors by constructing compound matrix and common group model, and integrating the detection results *w. r. t.* different scenarios, thus allowing historical and peer baselines to work together. As for the unsupervised learning algorithm, we choose autoencoder as the baseline model, then additionally introduce a memory module based on attention weights to enlarge the reconstruction errors of anomalies. Note that the temporal and spatial statistics mentioned here are not actually related to time and space but an extension in a broad sense. Specifically, the temporal statistics not only denote the specific temporal information when user behavior occurs but also include the potential sequence relationships and variation tendency of activity frequencies. As for the spatial statistics, it refers to the correlations between peers' behaviors.

The main contributions of this paper are summarized as follows:

(i) We propose a novel user behavior temporal-spatial statistics fusion method to achieve the parallel historical and peer baselines. We perform comprehensive evaluations under different scenarios to obtain the final results. Our evaluation results clearly show the usefulness of MAITD.

(ii) We introduce a memory-augmented network into autoencoder to optimize the unsupervised learning algorithm. By enlarging the reconstruction errors of anomalies, it helps to reduce the false negative rate of detection. To the best of our knowledge, this is the first work to apply memory-augmented network on insider threat detection.

(iii) We perform extensive evaluations on CERT dataset [14] to demonstrate the superiority of our MAITD. The experimental results show that MAITD achieves the state-of-the-art performance on both instance-based and user-based settings.

The rest part of this paper is organized as follows: Section 2 summarizes the related work on insider threat detection, and outlines the differences between our approach and other similar works. Section 3 introduces the research motivation and basic idea. Section 4 presents the overall framework of MAITD and implementation details. Section 5 provides the detailed evaluation and analysis results. Section 6 discusses the limitations of MAITD and future work. Finally, Section 7 concludes this work.

2. Related Work

As an important component of system security protection, insider threat detection has attracted extensive attention from the research community. On the one hand, multiple insider-related projects like ADAMS, CINDER, and SCITE have been released successively by DARPA to prevent confidential data being stolen by insiders [15]. Among them, the latest work of SCITE project suggests that it is a feasible solution to detect insider threat by observing employee's reaction to tentative signals, which provides a new research strategy for reducing manual investigation burden [16]. The technical report released by CERT Insider Threat Center records various practical cases and corresponding mitigation

and preventive measures [17]. On the other hand, there are also many excellent surveys and solutions in the academia. Liu et al. [5] and Homoliak et al. [18] focus on the definition, taxonomy, and categorization of insider threats, and give a detail review of current research situation. Yuan et al. [19] discussed the opportunities and challenges of insider threat detection in the era of deep learning. The threat description models proposed by Pfleeger et al. [20] and Nurse et al. [21] believe that insider threat is the result of interaction of system environment, personal character, and historical behavior. However, it should be noted that the discussion of behavioral models that attempt to correlate insider threats with psychological profiles of users are outside the scope of this paper. Here, the insider threat detection is defined as: "At any given time instance, given their past online activities, how to predict if an employee is behaving abnormally either with respect to his past activity, or with respect to the behavior of his peers" [8]. That is, we simplify insider threat detection as anomaly behavior detection, and focus on how to effectively exploit the temporal and spatial characteristic of user behavior to improve detection performance. In view of the important impact of behavior representation and detection algorithm on solution performance, we will introduce the related studies from the perspective of temporal-spatial characteristic utilization and unsupervised detection algorithm optimization.

2.1. Temporal-Spatial Characteristic Utilization. Most insider threat detection schemes are based on historical baseline or peer baseline, in which the former represents the past habitual pattern of individual user while the latter focuses more on behavioral correlation between members in the same group. Normally, they were not in conflict but complementary, each with its own sphere of competence. However, most solutions only focus on one side (especially historical baseline), leading to the limited detection performance [8]. In order to more clearly elaborate the current research, we made a more fine-grained partition on the basis of previous works. Firstly, in the context of historical baseline schemes, some works tend to capture the temporal characteristics directly by means of the models with temporal learning capabilities. Examples of such models are statistical models [6, 8, 9, 22–24], Hidden Markov Model [11], Graph Embedding Model [25], Recurrent Neural Network [10, 12, 26], and Self-Attention Mechanism [27]. Gavai et al. [8] propose to capture time-varying characteristics by taking a weighted average of activity frequency feature to improve detection performance. Similarly, Chattopadhyay et al. [9] used the temporal indicators such as "Katz fractal dimension" and "total power corresponding to the top five frequencies in the power spectrum of the time-series signal" to generate time-series vectors, and combined with cost-sensitive undersampling technique to detect insider threats. In order to detect the low-signal yet long-lasting threats, Yuan et al. [6] constructed the compound behavioral deviation matrix to represent user behaviors. Different from the above schemes, Duc et al. [22–24] pay more attention on the impact of using the recent time window as a baseline comparison for each data instance, instead of designing new behavior representation. Excluding statistical methods, machine learning techniques have also been applied to building historical baseline models. Rashid et al. [11] used activity sequences as input, the hidden Markov model as modeling approach, and the deviation between predicted results and actual activities as judging criteria to detect anomalies. Liu et al. [25] developed an efficient anomaly detection system based on the graph embedding technique. Considering the powerful representation ability of deep learning models, Tuor et al. [12] and Sun et al. [26] used deep recurrent networks to detect insider threats. To further improve the accuracy of behavior model, Yuan et al. [10] combined the temporal point process with Long Short Term Memory (LSTM) to learn user's normal behavior pattern from four aspects: *activity duration*, *activity type*, *session duration,* and *session interval*. Inspired by position encoding [28], Yuan et al. [27] retained the absolute time information of user activities by calculating the minute offset, and used self-attention mechanism to construct the final behavior representation. Secondly, there are some other historical baseline schemes [13, 29, 30] that prefer to capture temporal characteristics in a round-about way. In these schemes, the model itself is only a modeling method, and training data and training mode are key. In other words, they simply take the individual historical behaviors as the model input. For example, Liu et al. [13] used the "4W" template to reorganize audit logs, and arranged them based on user id in chronological order to form training corpus. This indirect extraction method has the advantage of simplicity, but also confronts the challenge of limited performance.

Compared with plentiful historical baseline schemes, there are a few researches on peer baseline. One possible reason is that it is not easy to define the boundary of peers. Generally speaking, by comparing the difference between individual behavior and his peers' behavior, insider threat detection schemes can reduce the false positive rate in occasions (e.g. service outage and environmental change) where many users have common burst of events. The peers here do not simply refer to those users in the same department, but groups with similar behavioral trends. There are two broad approaches to divide the peers: role-based division [6, 31–33] and cluster-based division [34, 35]. The former arranges users into groups according to their roles, while the latter classifies users by clustering their behavior features. For example, Eldardiry et al. [34] divided users into different peer groups by calculating the similarity between behavioral data. Another issue closely related to peer baseline is how to represent the group's behavior pattern. A common solution is to extract behavior features automatically with the help of neural network model, and the key is that the training samples for network model should be the behavioral data of peer group rather than individual data [32, 36]. In addition, Yuan et al. [6] and Gavai et al. [8] used the statistical average to build peer baseline, but their implementation details are different. To sum up, the behavioral baseline model is closely related to the training mode, and how to effectively represent user behavior is still a problem worth exploring.

2.2. Unsupervised Detection Algorithm Optimization. Since the practical behavioral logs do not contain label information, unsupervised detection methods are the current mainstream study direction. Many classical unsupervised learning algorithms such as K-means Clustering [34], Isolated Forest (IF) [8] and One-class Support Vector machines (OCSVM) [37] have been applied in the field of insider threat detection. However, due to user activity spreading across multiple behavior domains (i.e. complexity), the above methods always achieved suboptimal performance. As a typical representative of reconstruction-based methods, autoencoder is favored by security practitioners because of its robustness on domain knowledge and stronger anti-interference ability. Briefly speaking, an autoencoder-based anomaly detection model only learns how to reconstruct normal samples, so the reconstruction error becomes higher for the abnormal samples. For example, Yuan et al. [6] and Liu et al. [38, 39] used fully connected autoencoders to learn normal behavior pattern, and regarded those whose reconstruction errors exceed predefined threshold as anomalies. However, although these reconstruction-based approaches have achieved fruitful results, there is still much room for improvement.

In order to improve the detection performance of reconstruction-based schemes, researchers have successively proposed various optimization methods. Zhou et al. [40] proposed a robust autoencoder model based on alternative optimization to strengthen the anti-noise capacity. Zong et al. [41] used a deep autoencoding Gaussian mixture model to detect anomalies, and optimized model parameters in an end-to-end way. Nguyen et al. [42] applied the variational autoencoder to anomaly detection, and attempted to explain anomaly from the aspect of gradient descent. Gong et al. [43] used memory-augmented network to optimize anomaly detection scheme, and achieved good results in multiple datasets. On this basis, Park et al. [44] further improved performance by introducing extra loss functions such as intra-class distance and inter-class distance, but its application scope was limited to the computer vision field. Rather than attempting to optimize model architecture, Mirsky et al. [45] chose to use multidetector integration to improve detectability. Besides, Yuan et al. [46] discussed the impact of different reconstruction methods (i.e. single-event prediction or sequence recomposition) on anomaly detection performance. Note that, although these works provide fruitful insights on optimizing unsupervised detection algorithm, it is difficult to apply their model directly on insider threat detection due to the requirement difference for the input data. Hence, how to choose the optimal unsupervised detection algorithm according to application scenarios is an open problem.

Compared with the existing works, our MAITD mainly focuses on the problem of temporal-spatial characteristics fusion, and at the same time chooses the memory-augmented autoencoder as an unsupervised detection algorithm to improve performance. In this regard, Acobe proposed in work [6] is similar to our MAITD, but its compound behavioral deviation matrix loses some critical behavior change information. Specifically, the main differences between MAITD and Acobe are as follows: (i) In terms of temporal characteristic analysis, Acobe generates the compound behavioral deviation matrix by concatenating multiple consecutive single-day feature vectors, while MAITD adds temporal indicators behind basic features to generate specific input for the temporal representation model. In other words, the compound matrix of MAITD contains the initial frequency information in addition to behavior variation information, so it has stronger representational capacity. (ii) In the aspect of spatial characteristic analysis, Acobe uses the same extraction method as temporal characteristics (that is, adding group feature to the compound deviation matrix), while MAITD builds an extra common group model to capture spatial characteristics. (iii) In terms of temporal-spatial fusion, the weighted summation mechanism gives MAITD more flexibility because it can adjust the weight factor according to application scenario, but Acobe is relatively rigid because it uses a single behavior model to simultaneously capture temporal and spatial characteristics. (iv) Unlike Acobe with full-connected autoencoder, MAITD chooses the memory-augmented autoencoder as unsupervised detection algorithm to improve performance.

3. Motivation

Although anomaly detection system is typically not used as a standalone solution, it plays an important role in assisting security analysts in selecting suspicious activities to be further scrutinized. However, the existing solutions are unable to satisfy growing practical demand, and how to improve insider threat detection performance has become a common goal in the research community. Driven by this goal, we first summarize the factors that affect detection performance in combination with application scenarios, and then propose the corresponding improvement measures for the existing problems.

Considering that false negatives and false positives are two critical evaluation metrics of anomaly detection system, we intend to elaborate the existing drawbacks from two aspects: malicious activities that are difficult to identify and normal activities that are easy to misidentify. First of all, some malicious activities will not be completed quickly in a short term, but there will be a long process of "commission." For example, in order to avoid exposure while stealing confidential data, the long-dormant spy usually does not steal numerous confidential documents at once but leaks sensitive data piece-by-piece in a long run. This means that the above threat scenario does not cause immediate behavioral deviation, and only long-term monitoring and analysis of user behavior can detect them. Many solutions [29, 30, 38, 39] only focus on the single-day and individual user behavior features, without considering the changes over multiple consecutive days, making it difficult to detect low-signal but long-term threats. Even though there are some works [6, 8, 9, 22, 24] in exploring the variation tendency of user behavior, they have different limitations, respectively. These schemes either design ideal—overly optimistic—time-varying indicators [9] or have trouble in keeping balance

between precision and overload [6]. Moreover, for autoencoder-based detection schemes, the assumption that anomaly incurs higher reconstruction error does not always hold in practice since those anomalies similar to normal activities can also be reconstructed well [43]. For example, when activity frequency is used as baseline model input, the number of connections to removable devices in malicious scenario (such as data breach) is similar to the frequency in normal scenarios (such as data migration result from device updates), which will make it difficult to distinguish whether the higher reconstruction error is from normal sample or anomaly.

To mitigate the above drawbacks, we plan to reduce the false negative rate of detection scheme from two perspectives. On the one hand, we hope to design more reasonable behavior representation to capture multidimensional temporal characteristics, and at the same time take the difference between different types of behaviors into account. This requirement inspires us to design new temporal indicators while retaining original activity frequency information. On the other hand, we intend to optimize the unsupervised detection algorithm to improve performance. It has been suggested that the memory-augmented network is beneficial to enlarge the reconstruction error of abnormal sample, so we plan to apply it in the field of insider threat detection.

Secondly, some normal behavior deviations can also be misidentified as anomalies. For example, due to the sudden service outage or environmental change, the employee's work pattern will inevitably change (such as longer work hours and more interaction, etc.), but these normal behavioral deviations cannot be correctly identified by the historical baseline model. This phenomenon fully indicates the importance of building a peer baseline. Generally speaking, there often exists certain behavioral correlation between an individual user and his peers, which provides a theoretical basis for the establishment of peer baseline [6, 8]. Given the probability that the whole group members are malicious is extremely small, we can make the following hypothesis: the greater behavioral correlation a user has with the group, the less likely the user is malicious. Based on this assumption, we plan to build an additional common group model to mitigate the above misreporting problem.

4. Methodology

The goal of this paper was to improve insider threat detection precision by analyzing the temporal and spatial characteristics of user behaviors. To this end, we propose a memory-augmented insider threat detection approach named MAITD. We first demonstrate the overall framework of MAITD, and summarize its basic idea and workflow. Then, we elaborate the temporal and spatial representation models and the improved unsupervised detection algorithm, respectively. Finally, we give the temporal-spatial fusion mechanism and corresponding implementation algorithm.

4.1. System Overview. Figure 1 shows the overview of MAITD, which mainly includes four modules: basic feature extraction, temporal characteristic analysis, spatial characteristic analysis, and comprehensive evaluation. The basic feature extraction module is responsible for multisource data collection and feature coding. Specifically, it extracts the behavior frequency features from multisource audit logs according to potential threat scenarios, and feeds them to the temporal and spatial characteristic analysis modules for further processing. In this process, the user's activities over a day are aggregated into a data instance to obtain better tradeoff between detection precision and response time. Based on these basic features, the temporal characteristic analysis module can calculate the temporal indicators according to the sliding window and predefined formulas, and then add them behind basic features to generate the compound behavioral matrix. With the help of historical compound behavioral matrixes and unsupervised detection algorithm, we can build an independent temporal representation model for each individual user. Similarly, the spatial characteristic analysis module will also build an additional spatial representation model for each group, but this model is shared with all the members within the same group. That is, the training space of spatial representation model is expanded from individual historical data to group's historical data, and at the same time the model input is changed from compound behavioral matrix to basic feature vector. After the temporal and spatial representation models are trained, we can obtain the anomaly scores of testing sample in historical and peer baseline, respectively. Finally, the comprehensive evaluation module integrates the above scores in combination with specific application scenarios to generate the final lists of anomaly instances and suspicious users.

Before getting into the details of this scheme, we will state the problem studied in this paper clearly and give the corresponding mathematical formulation. As mentioned above, the insider threat detection problem can be simplified as: "Given employee's past online activities, how to predict if an employee is behaving abnormally either with respect to his past activity, or with respect to the behavior of his peers." Let $\overline{X}$ be the space of all activities, and let $X \subseteq \overline{X}$ be the set of normal activities. Given a sample $S \subseteq X$, group affiliation P and employee's past normal activities, how to construct an unsupervised classifier $h_s(x)$: $\overline{X} \longrightarrow \{0, 1\}$ so that the formula $h_s(x) = 0 \Leftrightarrow x \in X$ is as valid as possible is the real crux of the matter.

Considering that the classifier is trained in an unsupervised manner and the method of providing only anomaly label is inconvenient to the subsequent investigation by security analysts, it is more preferable to generate an ordered list of suspicious users. Note that, the calculation of partial evaluation metrics such as detection rate and precision is closely associated with the organization's investigation budget. In practice, investigation budget represents the available human resources for analyzing the anomaly behavior instances, post-training of the detection system, and performing the necessary actions in response [22]. The more the investigation budget, the larger the range of the threshold that can be set. In addition, our analysis report will distinguish between anomaly instances and suspicious users to

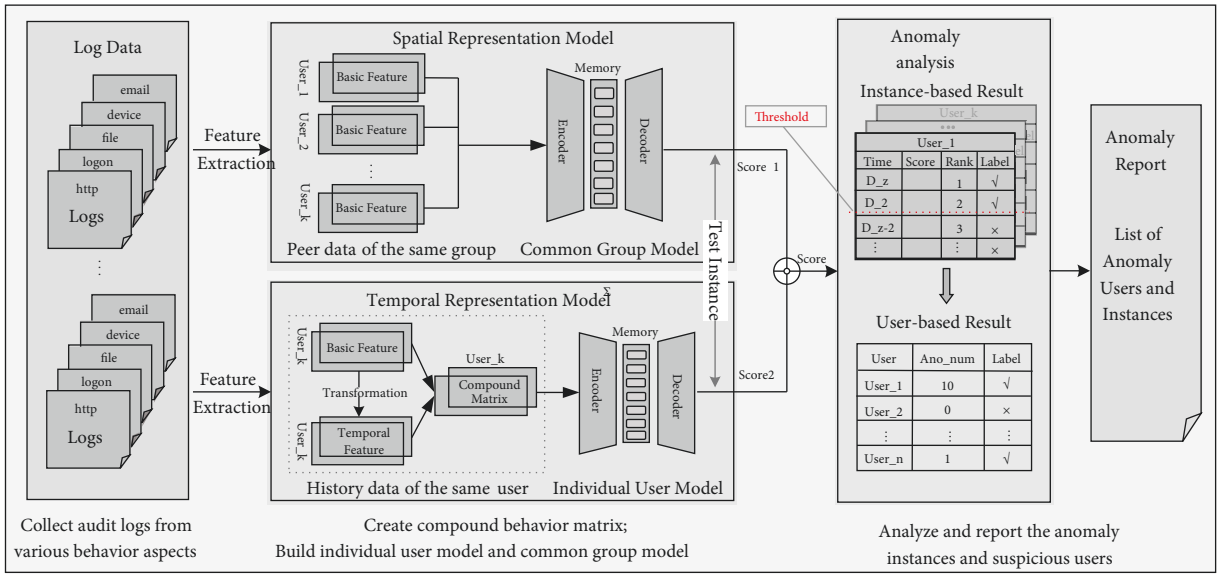


Figure 1: MAITD overview.

provide more comprehensive evaluation results. This is because high malicious instance detection rate is not synonymous with all malicious users being detected, and the latter is the ultimate goal of insider threat detection system. In this paper, suspicious users are defined as the users who have at least one anomaly behavior instances.

4.2. Temporal-Spatial Characteristics. The establishment of historical baseline and peer baseline is the key of insider threat detection scheme, directly influencing the quality of detection performance. In other words, how to make the best of the temporal and spatial characteristics of user behavior is a major challenge. To this end, our MAITD adopts two different methods to capture the temporal and spatial characteristics, respectively.

4.2.1. Temporal Characteristic Analysis. Inspired by Acobe [6], we find that the compound matrix is an effective way to capture the temporal characteristic. Because it can give the behavior model the ability to analyze temporal characteristics by adding time-varying elements to input. Motivated by this, MAITD also chooses the compound behavioral matrix as the input of the temporal representation model, but makes major changes in the element composition. As stated previously, the requirement that behavior representation not only contains multidimensional temporal information but also considers the difference between different types of behaviors gives us the inspiration to design new temporal indicators while retaining original activity frequency information. Therefore, we design a compound behavioral matrix with the structure shown in Figure 2(a) as the temporal representation model input. The compound behavioral matrix mainly consists of basic frequency features and time-varying features, and every feature can be further divided into two subparts (i.e. working hours 8 am to 6 pm and off hours 6 pm to 8 am) according to the occurrence time of user activity. In Figure 2(a), the basic feature set extracted in different behavior domains are arranged in the vertical direction. Given that the basic feature extraction is usually closely related to domain knowledge, here we take the CERT dataset as an example to design a series of basic features such as the number of copying file from other's PC during off hours and the number of visiting recruiting website on office computers during working hours. Here, note that the basic feature extraction is not the focus of this paper, and MAITD adopts the basic features proposed in our previous work, see literature [7] for details.

The horizontal direction in Figure 2(a) represents the different variants of basic behavioral features. The white area represents the normalized frequency information during working hours and off hours, while the blue area is filled with the feature variants (i.e. temporal indicators). As shown in Figure 2(b), the temporal indicators at the monitoring slot are calculated based on the historical values within the sliding window, and the reason behind it is that the sliding window mechanism ensures a smooth transition between the samples. Besides, we can also highlight that the decision of using all features from current and past observations is highly desirable since it may allow a near to optimal automatic weight assignment for past and current versions of each feature and provide a highly interpretable result [24]. In general, the compound matrix proposed in this paper does not simply concatenate multiple consecutive single-day features when capturing the temporal characteristics but relies on the property of temporal indicators. This is because simple data mergence cannot effectively reflect the variation tendency of user behavior but increase the unnecessary computational overhead [22]. Besides, a compound behavioral matrix represents the activity overview of a certain day, and "a certain day" here (e.g. the 5th day in Figure 2(a)) can be regarded as an index used to mark data instances.

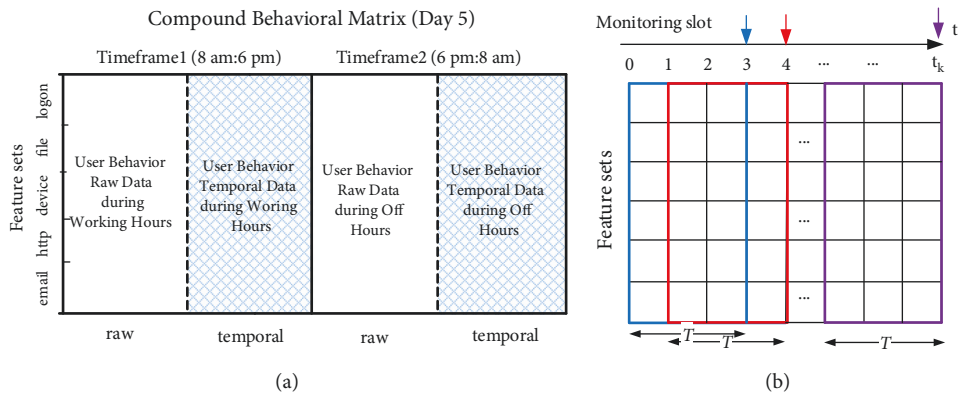


FIGURE 2: Compound behavioral matrix and sliding window mechanism. (a) The compound matrix can be divided into four main components according to activity timeframe and feature type. (b) The temporal indicators at the monitoring slot are calculated based on the historical values within the sliding window.

In fact, MAITD designs two different temporal indicators called standard factor and variation factor to capture the temporal characteristics of user behavior. The former mainly measures the relative size of user's basic features in the current time window, while the latter reflects the cumulative variation of the basic features within the current window. Let $g_{f,l,d}$ denote the numeric measurement of basic feature f in timeframe l on day d. $p_{f,l,d}$ and $q_{f,l,d}$ denote the standard factor and variation factor of basic feature f, respectively. $\overrightarrow{g}_{f,l,d}$ denotes the vector of history numeric measurements, and the detail can be expressed by (1), where l is the occurrence time of user behavior (0 or 1), T is the window size in days.

$$\overrightarrow{g}_{f,l,d} = \left[g_{f,l,i} | i\colon d - T + 1 \le i < d\right]. \tag{1}$$

Then, the standard factor $p_{f,l,d}$ and the variation factor $q_{f,l,d}$ can be calculated by the following transformation of $g_{f,l,d}$:

$$\begin{aligned} p_{f,l,d} &= \frac{g_{f,l,d} - \text{mean}\left(\overrightarrow{g}_{f,l,d}\right)}{\text{std}\, t\left(\overrightarrow{g}_{f,l,d}\right)}, \\ q_{f,l,d} &= (1-\beta) \cdot \sum_{j=1}^{T-1} \left(\beta^{T-j-1} \cdot \left(g_{f,l,d-T+j+1} - g_{f,l,d-T+j}\right)\right), \end{aligned} \tag{2}$$

where $\text{mean}(\overrightarrow{g}_{f,l,d})$ and $\text{std}(\overrightarrow{g}_{f,l,d})$ denote the mean and standard deviation of history measurements, respectively. β is the attenuation coefficient of feature variation, which is usually set as the reciprocal of window size. Actually, the standard factor can be regarded as the standardized transformation of basic feature within the current time window, and the variation factor is the exponentially weighted moving average of basic feature variation. Since the calculations of both indicators are closely relevant to the history measurements within the current window, they can represent the temporal correlation between user activities to some extent. Moreover, we set a lower bound for the standard deviation $\text{std}(\overrightarrow{h}_{f,l,d})$ to avoid worst cases where the standard factor is too large, and the related calculation equation is as follows:

$$\text{std}\left(\overrightarrow{g}_{f,l,d}\right) = \begin{cases} \varepsilon, & \text{std}\left(\overrightarrow{g}_{f,l,d}\right) < \varepsilon, \\ \text{std}\left(\overrightarrow{g}_{f,l,d}\right), & \text{std}\left(\overrightarrow{g}_{f,l,d}\right) > \varepsilon. \end{cases} \tag{3}$$

In addition to designing new temporal indicators, MAITD also takes the difference between different types of behaviors into account. In other words, we assign different weights for basic features and their variants to make the behavior model pay more attention on those features that are most helpful to improve the detection performance. In this regard, we use the same weight setting as Acobe scheme, that is, the weights are lower for chaotic features but higher for consistent features. The specific calculation method is shown in:

$$w_{f,l,d} = \frac{1}{\log_2\left(\max\left(\text{std}\left(\overrightarrow{g}_{f,l,d}\right), 2\right)\right)}. \tag{4}$$

After obtaining the weighted feature values, we should normalize the compound matrix to eliminate the adverse effects caused by different orders of magnitude. Finally, the temporal characteristic analysis module takes the compound behavioral matrix as input, the improved autoencoder as detection algorithm to build an independent temporal representation model for each user, thereby obtaining the anomaly score of behavior instance in historical baseline.

4.2.2. Spatial Characteristic Analysis. Different from temporal characteristic analysis, MAITD does not use the same way to capture spatial correlation between user behaviors, and instead achieves this goal by building an additional common group model. In short, the spatial characteristic analysis module first divides users into groups according to their roles, and then takes the basic feature vectors as input, the improved autoencoder as detection algorithm to build spatial representation model. It should be noted that this model is shared with all the members within same group. Since the group model captures the spatial characteristics based on neural network itself, it can be regarded as an end-to-end and data-driven method, which is similar to work [32, 36].

4.3. Detection Algorithm Optimization. As stated earlier, the autoencoder has become the mainstream insider threat detection algorithm because of its robustness on domain knowledge and stronger anti-interference ability. Therefore, this paper also chooses the deep autoencoder as the basic detector, and adds a memory module to improve detection performance. In general, the autoencoder consists of an encoder to obtain the compressed representation from the input and a decoder that can reconstruct the input purely based on the compressed representation. In the context of anomaly detection, the autoencoder is usually trained by minimizing the reconstruction error on the normal samples, and then uses the reconstruction error as an indicator of anomalies. Since the autoencoder only learns how to reconstruct the seen normal behaviors, those poor reconstructions result from behaviors that have not yet been seen. However, this does not mean that all anomalies can be detected effectively, because those abnormal samples that have many similarities with normal behaviors can also be reconstructed well. Inspired by work [43], we augment the deep autoencoder with a memory module based on attention weight to enlarge the reconstruction error of abnormal samples, thus achieving the goal of reducing the false negative rate. The intuition behind introducing a memory module is that there is a larger differential between the abnormal sample and the sample reconstructed from the normal behavior representations. With that in mind, we apply memory-augmented network to insider threat detection problem, and propose the improved unsupervised detection architecture shown in Figure 3.

Compared with the traditional autoencoder, the improved autoencoder (i.e. Autoencoder-Mem) adds a memory module between the encoder and the decoder to reprocess the compressed representation. In a way, the memory module can be regarded as a storage component used to record prototypical normal behavior patterns. It is through this component that MAITD can map the abnormal samples to the most relevant normal behavior patterns for reconstruction, resulting in an output significantly different to the anomaly input. Based on this mechanism, the reconstruction errors of abnormal samples can be further enlarged. Specifically, given an input x, the encoder Φ first obtains the initial compressed representation $\Phi(x)$. By using the compressed representation $\Phi(x)$ as a query, the memory module retrieves the most relevant items m_i in the memory M via the attention-based addressing operator [43] to generate the reprocessed representation $\Phi(x)'$, which is then delivered to the decoder for reconstruction. After decoding the reprocessed representation $\Phi(x)'$, the decoder Ψ can obtain the reconstructed sample $\hat{x}$, and calculate the mean square error between $\hat{x}$ and x as the model output. In this process, the essence of mapping is to reconstruct the compressed representation based on the prototypical normal behavior patterns recorded in memory, and it is actually realized by using attention-based memory addressing. Let query z denote initial compressed representation $\Phi(x)$, M denote the memory network with prototypical normal behavior patterns, and w denote the attention weight row vector of each item in M for the query z. Then, the weight entry w_i of w and reprocessed query $\hat{z}$ can be obtained by the following equations:

$$w_i = \frac{\exp\left(zm_i^T / \|z\| \|m_i\|\right)}{\sum_{j=1}^{N} \exp\left(zm_j^T / \|z\| \|m_j\|\right)}, \quad \forall i \in \{1, 2, \ldots, N\}. \tag{5}$$

$$\begin{aligned} &\hat{z} = w \cdot M = \sum_{i=1}^{N} w_i \cdot m_i, \\ &\text{s.t.} \quad \sum_{i=1}^{N} w_i = 1, \end{aligned} \tag{6}$$

where row vector m_i is any item of memory network M, representing a prototypical normal behavior pattern. The dimension of item m_i is same to query z, and N is the capacity of the memory network M.

In addition, we also applied the sparse addressing method proposed in Ref. [47] when reconstructing the query z to make the memory network learn more accurate normal behavior patterns:

$$\widetilde{w} = \frac{\overline{w_i}}{\|\overline{w}\|_1}, \overline{w_i} = \frac{\max(w_i - \lambda, 0) \cdot w_i}{|w_i - \lambda| + \epsilon}, \tag{7}$$

where λ is the lower bound of the weight m_i in the sparse addressing process, and ϵ is a very small positive scalar to avoid divide-by-zero exception. After finishing the above sparse addressing operation, we can obtain the reprocessed query $\hat{z}$ based on (6). Simply speaking, the sparse addressing encourages the model to represent an example using fewer but more relevant memory items, leading to learning more informative representations in memory.

Due to the introduction of the memory module, we adjust the objective function used in the training process. In addition to minimizing the reconstruction error on each sample, we take the sparsity characteristic of memory-augmented network into account, and add the sparsity regularizer on attention weight $\widetilde{w}$. The final objective function is shown in (8), where K is the size of the training set, and α is a hyper-parameter in training. Note that, despite using a new objective function in the training stage, we still use the l_2-norm based mean square error, i.e., score $= \|x - \hat{x}\|_2^2$, to measure the anomaly score of test sample.

$$\text{Loss} = \frac{1}{K} \sum_{i=1}^{K} \left(\|x_i - \hat{x}_i\|_2^2 - \alpha \cdot (\widetilde{w}_i \cdot \log(\widetilde{w}_i)) \right). \tag{8}$$

The basic explanation of this optimization mechanism can be summarized as follows: during training, the encoder and decoder are dedicated to minimizing the reconstruction error, and the memory module is simultaneously updated to record the prototypical normal behavior patterns. At the test stage, the learned parameters of encoder, decoder, and memory module are fixed, and the reconstruction is obtained from a few selected memory items of normal behaviors. Thus, the reconstruction tends to be close to the normal sample, resulting in larger errors in abnormal behavior instances.

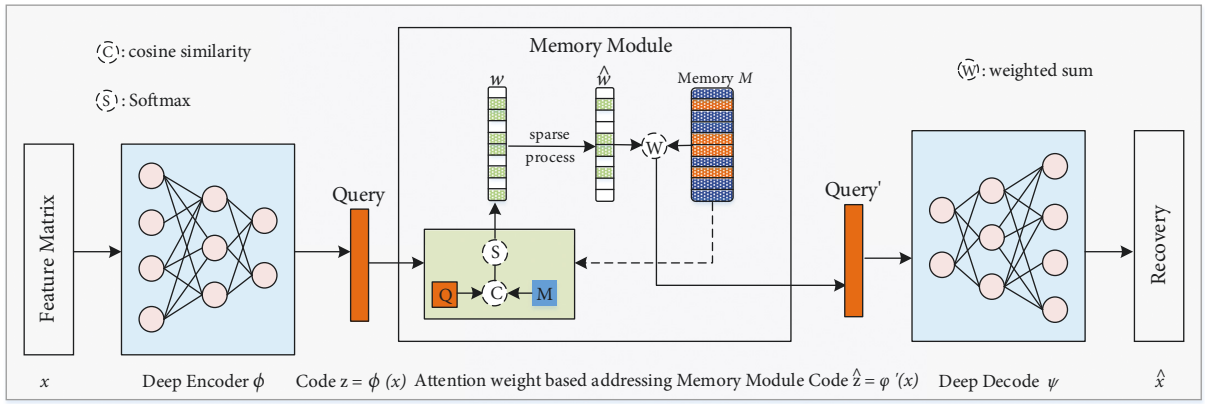


Figure 3: The improved anomaly detection model architecture.

4.4. Fusion Analysis. After building the temporal and spatial representation models, we can obtain the anomaly scores of test sample in historical and peer baseline, which are denoted by sco_t and sco_s, respectively. Subsequently, the comprehensive evaluation module calculates the final anomaly score of test sample by integrating the above detection results, and generates the final lists of anomaly instances and suspicious users according to whether it exceeds the predefined threshold. As for the specific fusion method, MAITD chooses the classical weighted summation way:

$$\text{sco} = \xi \cdot \text{sco}_s + (1 - \xi) \cdot \text{sco}_t, \tag{9}$$

where ξ is a hyper-parameter, representing the proportion of the spatial characteristic analysis module in the whole detection model. As mentioned previously, the setting of threshold is closely associated with the organization's investigation budget. In this paper, we adopt the following decision strategy:

$$\text{thre} = \text{mean}_{\text{train}} + \sigma \cdot \text{std}_{\text{train}}, \tag{10}$$

where $\text{mean}_{\text{train}}$ and $\text{std}_{\text{train}}$ denote the mean and standard deviation of anomaly scores of training samples, respectively. σ is the system input (i.e. investigation budget), which is set to 3 in this paper. Based on the above mechanism, MAITD can obtain the list of anomaly instances for each user, and label the users who have at least one anomaly behavior instance as suspicious users. Algorithm 1 shows the details of the whole insider threat detection scheme. Specifically, after the initialization work is completed, we can build the temporal representation model by constructing compound matrix and training the memory-augmented autoencoder (lines 2–10). Then in line 11, we split users into different groups according to group affiliation **P**. Next, the spatial representation model is built based on the basic features of group members (lines 12–22). Finally, our comprehensive evaluation module can generate the final anomaly scores of test samples according to the fusion mechanism and report the suspicious behavior set Λ and suspicious user set $\mathbf{U_a}$ (lines 23–33).

5. Evaluations

To verify the effectiveness and feasibility of MAITD, we performed extensive experiments on the CERT insider threat dataset [14]. We first introduce the dataset, evaluation metrics, and the experimental setting used in this paper and then compare MAITD with other representative schemes in detail. Next, we analyze the effectiveness of spatial-temporal fusion mechanism and the improved anomaly detection algorithm. Finally, we discuss the impact of the parameters on the detection performance.

5.1. Dataset. The CERT dataset released by Carnegie Mellon University is the most widely used public dataset in the field of insider threat detection. It contains multiple versions that simulate the daily behaviors of internal employees in different organizations. In this paper, we choose the latest r6.2 release as the primary dataset to evaluate the detection performance of MAITD, and at the same time use the classical r4.2 release as the secondary dataset to verify its generalization performance. These two datasets record the daily behaviors of 4000 and 1000 employees of different organizations within 516 days, and provide 5 predefined insider threat scenarios as detection objects. Specifically, these activities cover five behavior domains: logon, device, file, http, and e-mail, and the threat scenario can be regarded as a specific combination of the above activities. Since the malicious activities are usually rare in the real world, the class-imbalance problem is also embodied fully in these datasets. Such a phenomenon explains why supervised classification methods are not suitable for insider threat detection to a certain degree. Besides, due to the excessive overhead of processing the entire dataset (200G), we select several user groups to form a subset to conduct performance evaluation. During this process, in addition to the necessary groups of anomaly users, we also randomly select multiple groups without anomaly users to simulate the class-imbalance situation. Table 1 lists the main information of the dataset used in this paper.

Like most insider threat detection schemes [6, 9, 22, 39], we chose the following evaluation metrics, namely, detection

(i) Input: user set $\mathbf{U}$, behavior instance set Ω, group affiliation $\mathbf{P}$, threshold σ
(ii) Output: suspicious behavior set Λ , suspicious user set $\mathbf{U_a}$
(1) $\varnothing \leftarrow \Lambda$, $\varnothing \leftarrow \mathbf{U_a}$, $\varnothing \leftarrow \Omega_{\text{train}}^{\text{group}}$
(2) for $u \in \mathbf{U}$ do
(3) $\quad \Omega^u = \Omega_{\text{train}}^u + \Omega_{\text{test}}^u$
(4) $\quad$ for $x \in \Omega_{\text{train}}^u$ do
(5) $\quad\quad \widehat{x_t^u} \leftarrow x_t^u \leftarrow x$ //calculate the basic feature x_t^u and compound matrix $\widehat{x_t^u}$
(6) $\quad$ end for
(7) $\quad$ while not converged do
(8) $\quad\quad$ train the memory-augmented temporal model Model_t on $\widehat{x_t^u}$
(9) $\quad$ end while
(10) end for
(11) $\mathbf{G} \leftarrow \mathbf{U}, \mathbf{P}$ //split users into different groups according to group affiliation $\mathbf{P}$
(12) for group $\in \mathbf{G}$ do
(13) $\quad$ for $u \in$ group do
(14) $\quad\quad \Omega_{\text{train}}^{\text{group}} = \Omega_{\text{train}}^{\text{group}} \cup \Omega_{\text{train}}^u$
(15) $\quad$ end for
(16) $\quad$ for $x \in \Omega_{\text{train}}^{\text{group}}$ do
(17) $\quad\quad x_t^{\text{group}} \leftarrow x$ //calculate the basic feature vector x_t^{group}
(18) $\quad$ end for
(19) $\quad$ while not converged do
(20) $\quad\quad$ train the memory-augmented spatial model Model_s on x_t^{group}
(21) $\quad$ end while
(22) end for
(23) for $u \in \mathbf{U}$ do
(24) $\quad \text{thr} \leftarrow (\text{sco}_{\text{train}}, \sigma)$
(25) $\quad$ for $x \in \Omega_{\text{test}}^u$ do
(26) $\quad\quad \text{sco}_s \leftarrow (\text{Model}_s, x^{\text{group}}), \text{sco}_t \leftarrow (\text{Model}_t, x^u), \text{sco} \leftarrow (\text{sco}_s, \text{sco}_t)$
(27) $\quad\quad$ if sco > thr then
(28) $\quad\quad\quad \Lambda_u = \Lambda_u \cup \{x\}$
(29) $\quad\quad\quad \mathbf{U_a} = \mathbf{U_a} \cup \{u\}$
(30) $\quad\quad$ end if
(31) $\quad$ end for
(32) end for
(33) return suspicious behavior set Λ and suspicious user set $\mathbf{U_a}$

Algorithm 1: Memory-augmented insider threat detection approach with temporal-spatial fusion

rate (DR), precision (PR), F1-score, and the area under the receiver operating characteristic curve (AUC). Their calculation methods are as follows:

$$\begin{aligned} DR &= \frac{TP}{TP + FN}, \\ PR &= \frac{TP}{TP + FP}, \\ FPR &= \frac{FP}{TN + FP}, \\ F1 &= \frac{2}{PR^{-1} + DR^{-1}}, \end{aligned} \tag{11}$$

where true (false) positive (TP/FP) represents the number of malicious (normal) samples that are correctly recognized as "malicious," and false (true) negative (FN/TN) denotes the number of malicious (normal) samples that are incorrectly recognized as "normal." Among these metrics, AUC plays a more important role in evaluating solution performance because it is independent of the predefined threshold. In general, the larger the area under the curve, the better the performance of detection scheme. For the sake of brevity, all the performance metrics except AUC are reported in percent (%). Moreover, since the performance evaluation is reported in terms of both anomaly instance detection and suspicious user identification, there are two kinds of performance metrics: Instance-based (IDR, IPR, IF1, IAUC) and User-based (UDR, UPR, UF1, UAUC).

We implement the MAITD with Pytorch, in which both the temporal and spatial representation models adopt the architecture of 6-layer fully connected autoencoder plus a memory module. The related parameters are set as follows: the number of hidden units at each layer in the encoder and decoder are 256, 128, 64 and 64, 128, 256, respectively. The hyper-parameters $N, \lambda, \alpha, \epsilon$ in the memory module and ε in the compound matrix are, respectively, set as 100, 0.02, 0.002, 10^{-4}, and 0.01 according to the reference works [6, 43]. In addition, for each anomaly group, the training set includes the data from the first collection day until roughly one month before the date of the labeled anomalies, and the testing set includes the dates from then until roughly one month after the labeled anomalies. For normal groups, the user's behavior dataset is split into a training set and a testing

Table 1: Summary of dataset.

Dataset	Feature count	Mal_user: Nor_user	Mal_instances: Nor_instances
R6.2	112	5 : 812	45 : 51720
R4.2	112	70 : 866	966 : 55194

set in chronological order, and the splitting ratio is set to 30%. During training, we set the batch size and epochs as 32 and 40, and use Adam under the default parameters to optimize the detection model. All experiments are performed on compute nodes with Gold 5118 CPU, GTX 2060GPU, and 128 GB RAM, and the averaged results are reported after repeating 10 times.

5.2. Comparison with Other Works. In order to verify the superiority of MAITD, we compare it with other representative insider threat detection schemes. In this section, we select four similar works [6, 8, 9, 38] as comparison objects, and discuss their detection performance on the following three aspects: temporal representation model, spatial representation model, and the whole detection scheme. Among them, Acobe [6] is also designed to build the historical baseline and peer baseline simultaneously, but it adopts different spatial and temporal feature extraction methods (see related work for details). Gavai [8] and Pratik [9] design their own indicators to capture the temporal characteristics, respectively, and Liu [38] propose a simple autoencoder-based insider threat detection scheme. Prior to analysis, we first introduce the naming rules of experimental schemes to facilitate understanding. The name of experimental scheme consists of two parts: the former part "**" means the initial feature extraction method (i.e. basic features), and the latter part "##" represents the temporal or spatial representation model. For example, MAITD_Acobe_S denotes the detection scheme which adopts the MAITD's basic features and Acobe's spatial representation method.For the purpose of comparing temporal and spatial representation models, we design the following experiment scenarios. Under the conditions of same dataset, unsupervised detection algorithm, and parameter settings, we generate the comparative schemes by combining different basic features, temporal and spatial representation models, and then compare their performance to verify the superiority. Figure 4 shows the performance comparison results of different temporal and spatial representation methods on the r6.2 dataset. It can be seen from the left subgraph that the temporal representation method of MAITD has the best performance among all the temporal representation methods, and Acobe is better than Pratik and Gavai. This result further verifies the previous conclusion that simple data mergence cannot effectively reflect the variation tendency of user behavior. As for the poor performance of Partik and Gavai schemes, we think that the method of applying temporal indicators in other fields on insider threat detection directly is too ideal to get remarkable results. Likely, the experimental results in the right subgraph show that even if the basic features are changed, the common group model of TSDIM also performs better than the compound behavioral deviation matrix of

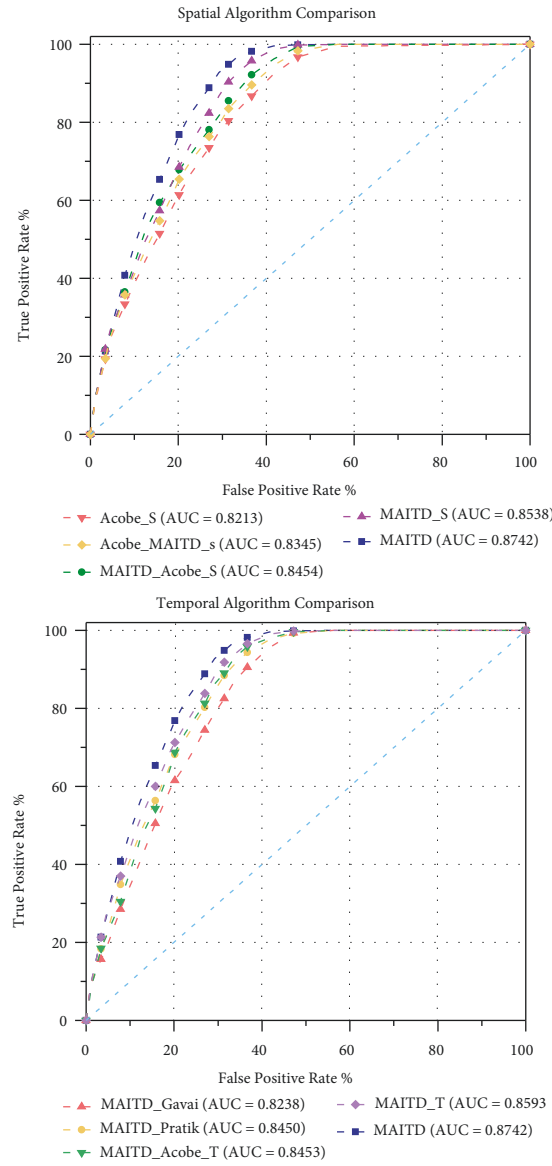


Figure 4: ROCs on r6.2 with different temporal and spatial representation methods.

Acobe in capturing the spatial characteristics. This is because Acobe only relies on the average of users' basic features in the same group when capturing the spatial characteristic, which loses much correlation information between peer's behaviors.Subsequently, we compare the whole detection scheme with other solutions. In addition to the recommended parameter setting, we also use some parameter tuning tools such as *hyperopt* [48] to optimize detection model when reimplementing comparative schemes. Figure 5 presents the experiment results based on the r6.2 dataset. As can be seen from this figure, MAITD outperforms other

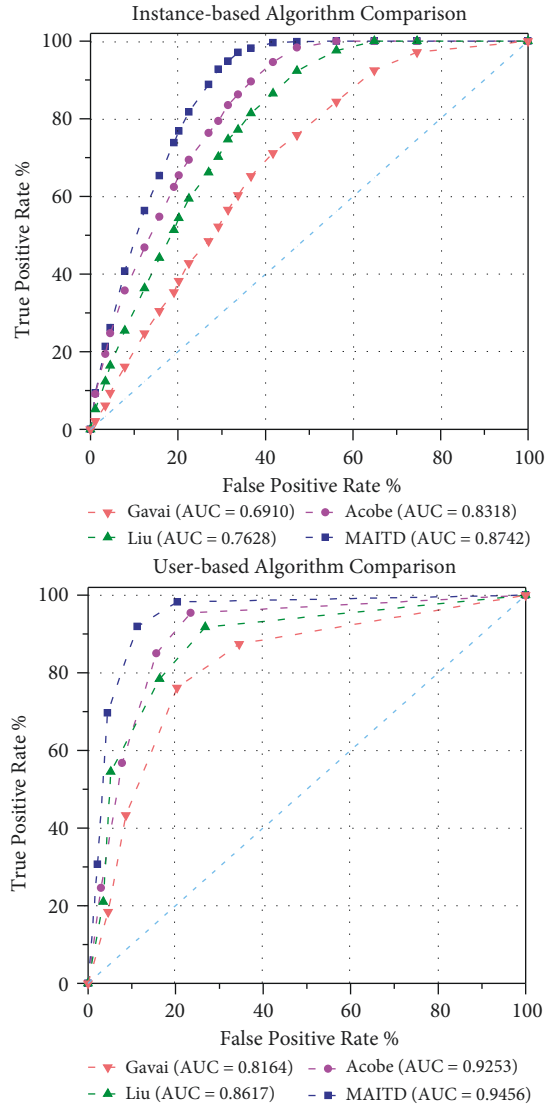


Figure 5: ROCs on r6.2 with different insider threat detection schemes.

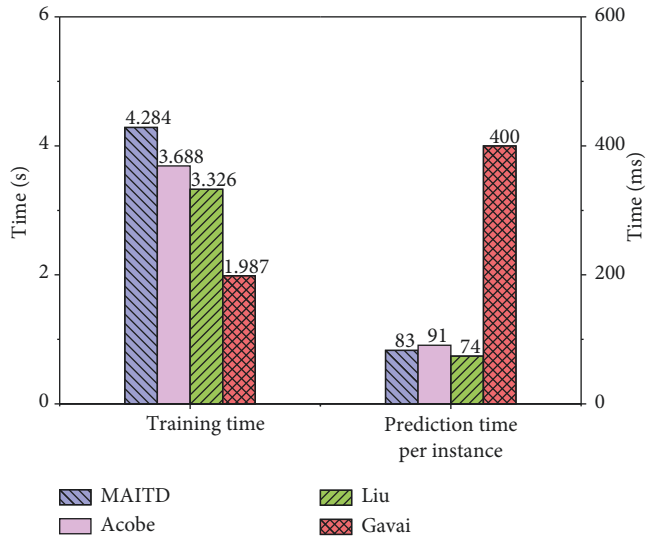


Figure 6: Average training time and prediction time per instance of different detection algorithms on r6.2.

three detection schemes in either case, and the instance-based and user-based AUC are, respectively, improved by 3.94% and 2.03% than the suboptimal scheme, which directly demonstrates the superiority of our scheme. Acobe gets suboptimal performance despite some defects in the aspect of temporal and spatial characteristic analysis. Surprisingly, the Gavai scheme with the ability of temporal characteristic analysis is weaker than the simple autoencoder-based scheme Liu. We think one possible reason is that the isolation forest used in Gavai is not suitable for insider threat detection. Because the success or failure of IF-based detection scheme is heavily dependent on the choice of good features and proper predefined contamination parameters, this prior knowledge usually is not known for security practitioners.

Moreover, we also make a comparison of model training time and prediction time per instance. As shown in Figure 6, the prediction time per instance of MAITD is shorter than Acobe despite opposite result in terms of training time. It can be explained through different baseline model construction methods, where MAITD adopts two independent representation models but others only build one individual model. But it should be noted that MAITD is not penalized in prediction time per instance as the individual and group models can be run in parallel. Besides, the size of the compound matrix of Acobe is significantly larger than MAITD, which invisibly introduces a large amount of computational overheads, thereby leading to longer prediction time. Although other schemes perform better than our TSDIM in training and prediction time, their poor detection performance also mean much human resources and additional investigation overheads. In general, our MAITD not only improves the insider threat detection performance but also takes into account the real-time requirement as much as possible.

Finally, to eliminate the adverse effects of accidental factors (such as the dataset is atypical or too small) on performance evaluation, we use r4.2 dataset to conduct the same comparative experiments. Compared with r6.2 release, r4.2 release has more positive samples and different organization background, so we think it is reasonable to verify the generality based on r4.2 release. More specifically, we record and compare multiple performance metrics to make a fair assessment, and the detailed information can be seen in Table 2 and Figure 7. It can be seen that the TSDIM scheme outperforms other methods even if the dataset used is changed. In addition, although the overall performance of MAITD on r4.2 is better than that on r6.2, the performance gap with other detection schemes is reduced. That is, we think that MAITD has more advantages in dealing with complex threat scenarios.

5.3. Ablation Study. In the previous section, we made a comprehensive comparison with other representative detection schemes. In the following, we will conduct several further ablation studies to verify the effectiveness of two

Table 2: The summary of insider threat detection results.

Data	Type	DR ($\sigma = 3$)				PR ($\sigma = 3$)				AUC			
		MAITD	Acobe	Liu	Gavai	MAITD	Acobe	Liu	Gavai	MAITD	Acobe	Liu	Gavai
r6.2	Instance	**69.06**	68.79	65.26	64.18	**50.34**	44.26	40.64	38.16	**0.8742**	0.8318	0.7628	0.6910
	User	**100**	**100**	80	80	**67.56**	63.15	59.91	47.43	**0.9456**	0.9253	0.8617	0.8164
r4.2	Instance	75.48	**75.61**	68.13	67.74	**53.72**	49.23	42.67	35.48	**0.8936**	0.8646	0.8127	0.7267
	User	**86.42**	83.36	80.62	72.64	**62.13**	59.86	55.37	49.35	**0.9551**	0.9434	0.9042	0.8673

The threshold used to calculate DR and PR is set as mean + 3*std, and the unit of DR/PR is percent. The bold values represent the maximum value i.e., the best performance among these methods.

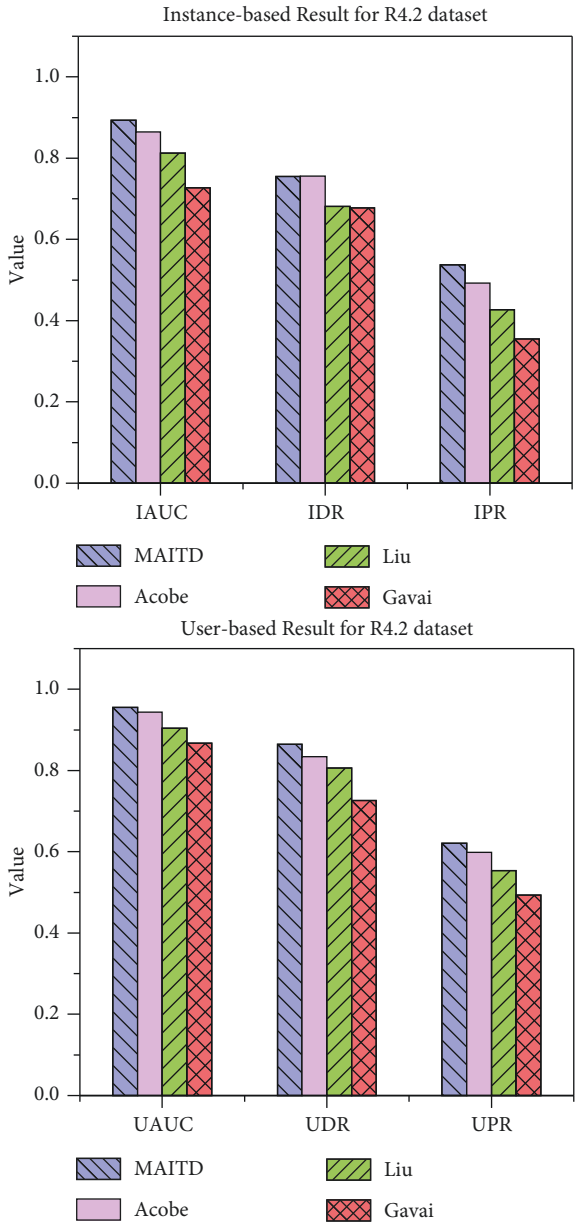


Figure 7: Performance comparison results of different insider threat detection schemes on r4.2.

optimization components. The naming rules of detection scheme are similar to the previous ones, but the latter part "##" can also represent the different unsupervised detection algorithms. For example, "MAITD-S″ refers to the detection scheme with the basic features and spatial representation model used in the MAITD method, and "MAITD-Vae" refers to the detection scheme which adopts the MAITD's temporal-spatial representation model and variational autoencoder-based detection algorithm.

The first experiment is used to evaluate the effectiveness of temporal and spatial characteristic analysis components. In this experiment, we choose the memory-augmented autoencoder as the unsupervised detection algorithm, and achieve the experimental goal by removing temporal or spatial representation models. Figure 8 records the instance-based and user-based ROCs on r6.2 dataset with different behavior representation models. It can be seen that removing either the temporal representation model or spatial representation model will degenerate the performance. Without the temporal (spatial) representation model, the insider threat detection scheme cannot capture the temporal (spatial) correlation between user activities, which may lead to the missing (false) alarms of the low-signal yet long-lasting threats (the collective behavior changes caused by occasional factors). Moreover, we also notice that the temporal representation model and the spatial representation model have the similar detection performance (the gap in IAUC is only 0.0055) when deployed separately, but there is still a certain gap (the gap in IAUC is 0.02) between them and the fusion scheme. This phenomenon indicates the necessity of temporal-spatial characteristic fusion in the field of insider threat detection. However, although the detection scheme with single representation model is worse than the fusion scheme, it performs better than the scheme without any representation model, thereby verifying the effectiveness of temporal and spatial characteristic analysis modules.

The second experiment is used to evaluate the effectiveness of the improved unsupervised detection algorithm. To have a fair comparison, all detection schemes leverage the temporal-spatial fusion component to capture potential correlation between user behaviors, and we choose two other detection algorithms as comparison objects. Here, we refer to the detection scheme with a fully connected autoencoder as Baseline, and generate new schemes by adding a variational mechanism (denoted as MAITD-Vae) and memory module (denoted as MAITD-Mem). Figure 9 shows the ROCs on r6.2 with different unsupervised detection algorithms. Experimental data show that the variational autoencoder provides limited performance improvement (0.0079), but the improvement brought from memory module (0.0236) is 3 times the former. This is because the variational autoencoder is designed to strengthen the ability to resist noise data, and it plays an important role in

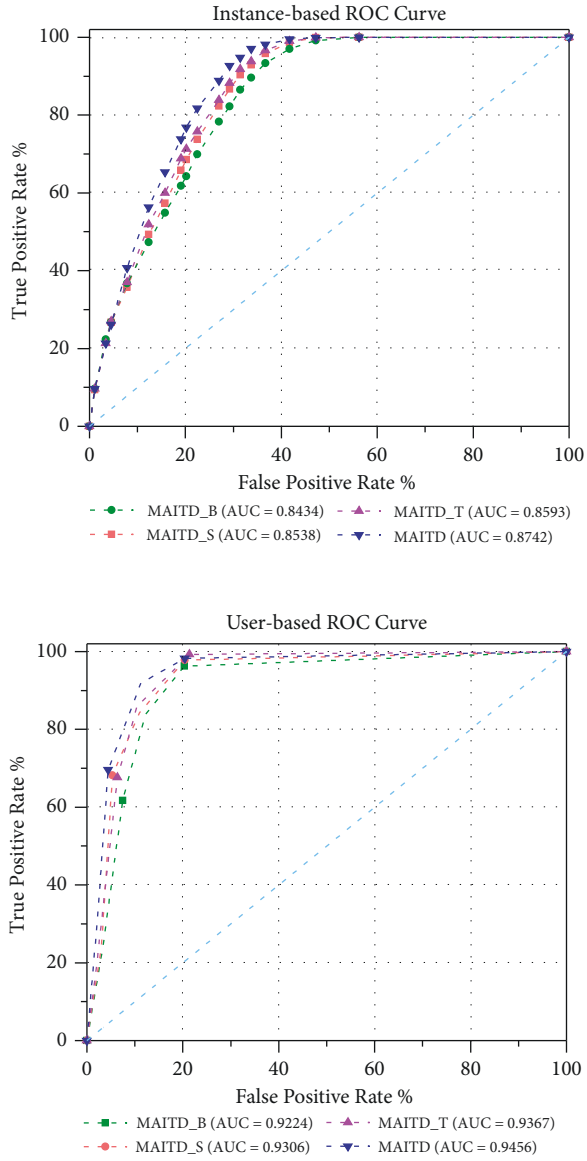


Figure 8: Rocs on r6.2 with different behavior representation models.

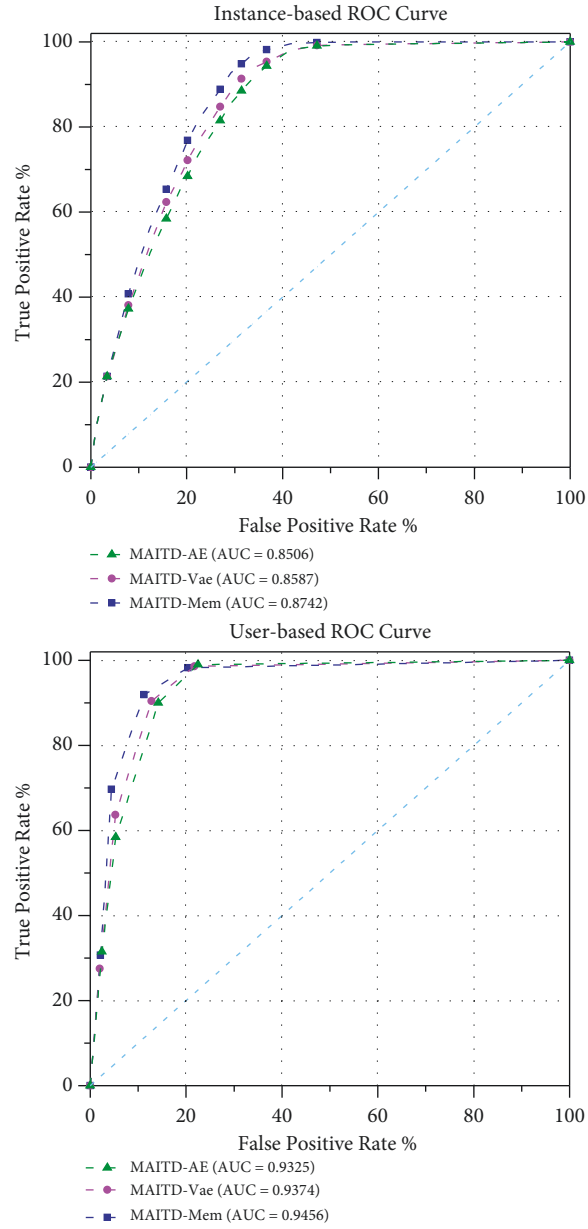


Figure 9: ROCs on r6.2 with different detection algorithms.

decreasing reconstruction errors of normal samples. Although this method is beneficial to distinguish anomalies in a way, it still faces the problem that some anomalies can also be reconstructed well. Instead, the memory-augmented autoencoder alleviates this problem by enlarging the construction errors of anomalies, which is more in line with the realistic demand of insider threat detection.

To explore the possible explanation for the optimization components, we analyze the trends of anomaly scores of two different users, and give the related case study. The detailed information can be seen in Figure 10, in which CMP2946 is a malicious user and LYB3419 is a normal user. The black curve denotes the anomaly scores of MAITD, and the gray curve denotes the anomaly scores of the MAITD-B scheme. The star markers at the bottom indicate the actual anomaly days. The false negative, false positive, and true positive are depicted by red, purple, and blue points, respectively. By comparing the number of red points in Figure 10(a), we can conclude that our two optimization components effectively reduce the false negatives in the detection results. Here, we take the behaviors of user CMP2946 on February 24, 2011 as a case, and briefly analyze the reasons for different results in two detection schemes. By studying the action sequence of the user on the day, we find that the number of visitors to the recruiting website is much less than the previous few days, and the number is even similar to the frequence during the normal period. This makes it difficult for detection scheme based on basic features to identify such anomalies. However, in addition to the initial frequence information, the temporal representation model of the MAITD scheme can capture the time-varying information hidden in user behaviors, so as to detect the above anomalies accurately. Furthermore, we can observe that the anomaly scores of anomaly user (CMP2946)

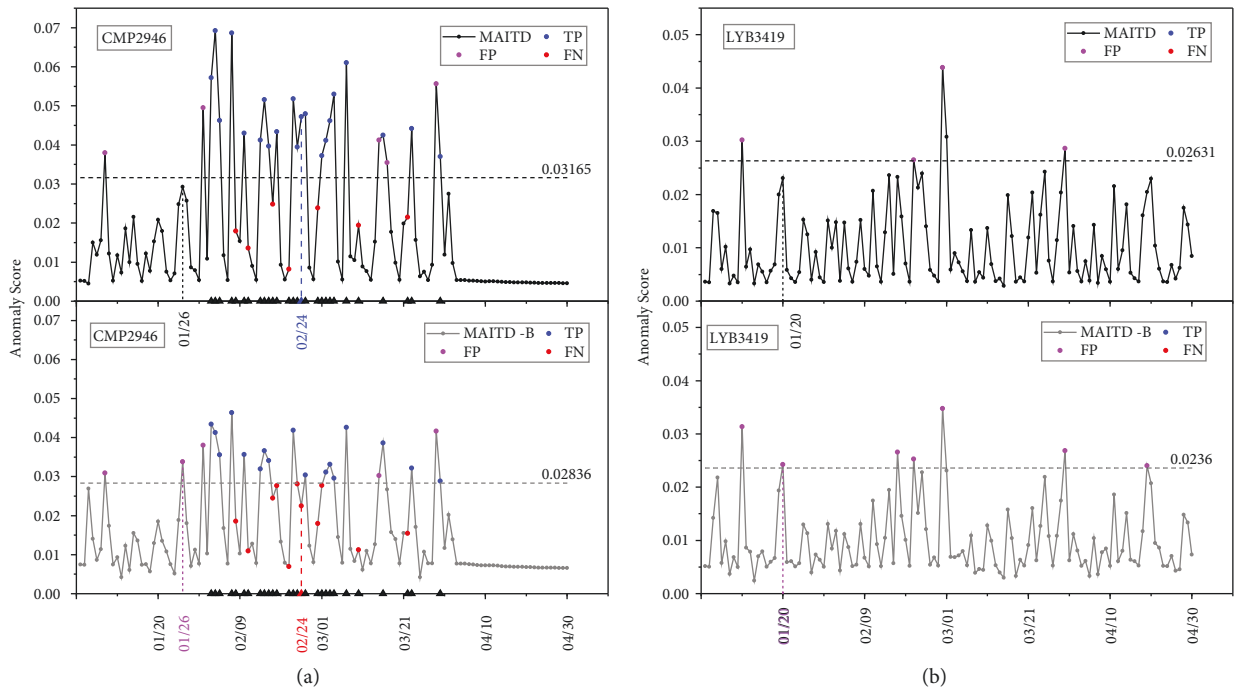


Figure 10: Trends of anomaly scores of different users on r6.2. (a) Malicious user CMP2946. (b) Normal user LYB3419.

in MAITD is obviously higher than that in MAITD-B scheme, while the normal user (LBY3419) has nearly the same anomaly scores. This phenomenon is consistent with the desired purpose of optimization components, thus verifying the feasibility of applying the memory-augmented network on insider threat detection.

Combined with the variation of purple points in Figure 10(b), we believe that two optimization components also play an important role in reducing the false positives. For example, two normal instances (the behaviors of user CMP2946 on January 26, 2010 and the behaviors of user LBY3419 on January 20, 2010) are successfully transformed from false positive samples to true negative samples. However, since there are no specific descriptions about occasional factors in the CERT dataset, we cannot conduct in-depth analysis to explain these phenomena. We think the application of spatial representation model is the main reason, and work [6] makes the same guess. In summary, the temporal-spatial fusion mechanism and the improved unsupervised detection algorithm are practical and feasible optimization measures.

5.4. Parameter Analysis. When describing the MAITD, we emphatically introduce two parameters, the size of sliding time window T and the model weight coefficient ξ, to help optimize the detection performance. The size of sliding window T is related to the scale of historical data used by the temporal representation model, and the weight coefficient ξ is responsible for adjusting the balance between historical baseline and peer baseline. Generally speaking, the larger the window size, the more historical information the temporal representation model can leverage, and the more advantageous for the detection of low-signal yet long-term anomalies. However, an overlarge window size will also weaken the short-term variation of user behavior, thereby lowering the ability to detect the sudden appearing threats. Meanwhile, there is no one-fit-all weight coefficient for every threat, and its value usually depends on the specific threat scenario. For example, in organizations with relatively obscure roles and functions, the behavioral patterns of members in the same group are not similar, so the importance of peer baseline in the whole detection system should be weakened. For these reasons, we prefer to obtain the best values of these parameters in the CERT dataset through numerical experiments.

Firstly, we evaluate the impact of window size T on the detection performance. Figure 11(a) shows that when other parameters are fixed, multiple evaluation metrics vary with different window size T. Note that the y-axis of Figure 11 represents the value of multiple evaluation metrics. It can be seen from the figure that the detection precision of anomaly instances and suspicious users shows an upward trend with the increase of window size, but this trend goes into reverse when the window size reaches a relatively large value. This phenomenon is line with our expectation, that is, the increase of window size expands the scale of historical data that the temporal representation model can leverage, but overlarge window size also hinders the acquisition of weak variation feature. Moreover, we also observed that other metrics such as detection rate show similar trend despite weak amplitude of variation, and it can be explained through the following two reasons. First, there are only a few anomaly instances and fewer malicious users in the dataset, which limits the variable range of evaluation metrics. Second, some rare anomalies are inherently difficult to detect based on

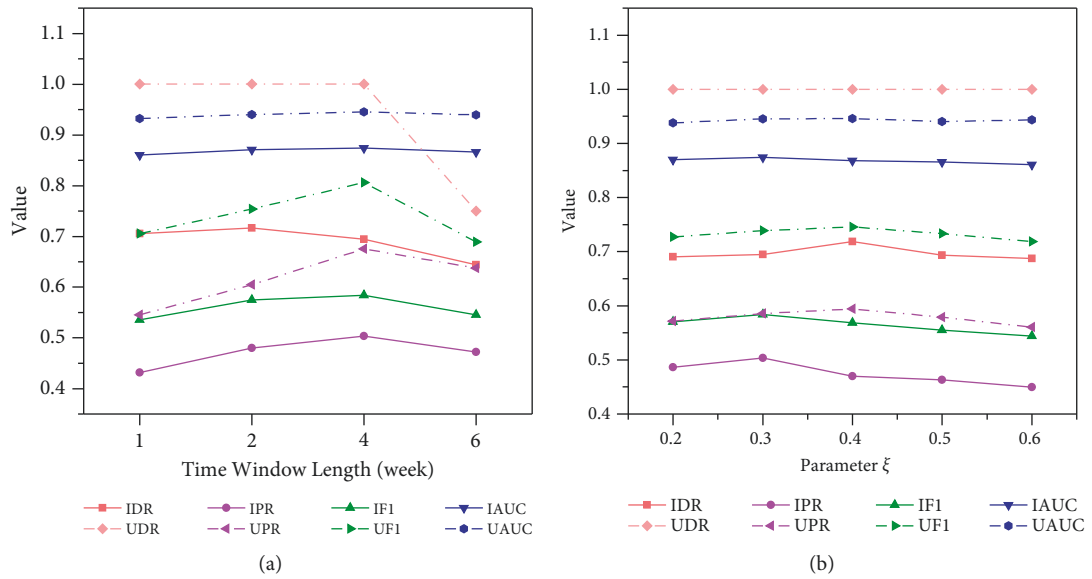


Figure 11: Parameter comparison results on r6.2. (a) The relationship between window size T and performance. (b) The relationship between parameter ξ and performance.

Table 3: The detection results of different weight coefficients.

Para ξ	Instance-based results				User-based results			
	IDR	IPR	IF1	IAUC	UDR	UPR	UF1	UAUC
0.1	69.04	48.61	57.05	0.8701	100	57.15	72.73	0.9382
0.2	69.46	**50.34**	**58.37**	**0.8742**	100	58.56	73.86	0.9456
0.3	**71.86**	49.01	58.28	0.8693	**100**	**59.45**	**74.57**	**0.9461**
0.4	69.34	48.31	56.95	0.8657	100	57.91	73.34	0.9406
0.5	68.74	45.98	55.10	0.8608	100	56.05	71.83	0.9438

The bold values represent the maximum value i.e., the best performance among these methods.

historical baselines. Therefore, we believe that the whole performance of insider threat detection scheme is at a relatively high level when the size of sliding time window is set to be 4 weeks (i.e. one month).

Secondly, we also design a numerical experiment to evaluate the impact of model weight coefficient on detection performance. Figure 11(b) and Table 3 show that when time size is set to be 4 weeks, the evaluation metrics vary with different weight coefficient ξ. It is observed that compared to the window size T, the impact of parameter ξ on detection performance is not so significant, but the variation trend of evaluation metrics is similar. Specifically, when the weight of the spatial representation model is at a relatively small level, increasing the value of parameter ξ is beneficial to improve the detection performance. But, it is undeniable that the temporal representation model plays a more important role in the whole detection process. From the data in Table 2, it can be concluded that when the weight coefficient ξ is set to be 0.3, multiple evaluation metrics are generally high.

6. Discussion and Future Work

Below we discuss limitations and future works. First of all, MAITD is still an insider threat detection scheme based on feature engineering in the traditional sense, and its many improvement measures are based on the premise of good basic features design. In other words, it is necessary for MAITD to enhance the detection ability of completely unknown threats, and this is also a common drawback of the traditional insider threat detection schemes based on feature engineering. To solve this, one option is to obtain the abstract representation of user behaviors by means of natural language processing technology. That is, we need to design a feature extraction scheme which can capture the potential semantic properties in the original audit logs without relying on any domain knowledge. Another possible solution is to encode discrete event logs into activity sequences, and build user's behavior profile to detect insider threat by utilizing the process mining method. Secondly, given that the MAITD detection model is trained on a fixed set of historical data, the online and system evolution in reality may cause significant performance degradation. Therefore, how to update detection model incrementally on newly arriving data to achieve consistently good performance with negligible cost is another important research direction. In this regard, Parveen [49] and Sun [26] provide an important reference to achieve this goal. Moreover, like most insider threat detection schemes, the results of MAITD lack intuitive interpretability and require further artificial investigation. In response to this problem, we believe that improving the

detection granularity of anomaly instances may be another feasible solution except for model interpretability study. In a way, the result itself has a certain interpretability when the detection granularity reaches the event level. We must admit that there is no one insider threat detection scheme which can detect all anomalies accurately without artificial investigation, and all that we can do is to reduce the investigation overhead as much as possible. In summary, although our MAITD suffers from some weaknesses, it provides an effective reference for other anomaly detection problems in the security field. Meanwhile, we will implement and verify the above possible solutions in future work, and positively contribute to a successful application of the proposed system in real-world scenarios.

7. Conclusion

Most existing insider threat detection schemes only focus on the historical behavior baseline while ignoring the peer baseline, resulting in poor detection performance. To solve this problem, we propose a novel insider threat detection scheme named MAITD, which adopts two different optimization measures to improve detection performance. First, it captures the temporal and spatial characteristics of user behaviors by constructing a compound behavioral matrix and common group model, and combines specific application scenarios to integrate the detection results, so as to enable both historical and peer baselines to work together. Second, it adds a memory module based on attention weight to autoencoder to enlarge the reconstruction error of the anomalies, and alleviate the false negatives. The experimental results on CERT datasets show that MAITD outperforms the latest insider threat detection scheme, and improves the instance-based and user-based AUC by 3.94% and 2.04%, respectively.

Data Availability

All the data used during the study were provided by Carnegie Mellon University CERT Insider Threat dataset. The dataset can be downloaded at https://kilthub.cmu.edu/articles/dataset/Insider_Treat_Test_Dataset/12841247/1.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported by a research grant from the National Science Foundation of China under Grant nos. 61772271 and 62106282.

References

[1] P. Institute, "2020 cost of insider threats global report," https://www.proofpoint.com/us/resources/threat-reports/2020-cost-of-insider-threats.

[2] Gurucul, "2020 insider threat survey report," https://gurucul.com/2020-insider-threat-survey-report.

[3] C. Insiders, "2020-cyber-threat-intelligence-report," https://www.cybersecurity-insiders.com/portfolio/2020-insider-threat-report-darktrace/.

[4] D. L. Costa, M. J. Albrethsen, and M. L. Collins, *Insider Threat Indicator Ontology*, Carnegie-Mellon Univ Pittsburgh Pa Pittsburgh United States, Pittsburgh, PA, USA, Tech. Rep, 2016.

[5] L. Liu, O. De Vel, Q.-L. Han, J. Zhang, and Y. Xiang, "Detecting and preventing cyber insider threats: a survey," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1397–1417, 2018.

[6] L.-P. Yuan, E. Choo, T. Yu, I. Khalil, and S. Zhu, "Time-window based group-behavior supported method for accurate detection of anomalous users," in *Proceedings of the 2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 250–262, IEEE, Taipei, Taiwan, JJune 2021.

[7] D. Li, L. Yang, H. Zhang, X. Wang, L. Ma, and J. Xiao, "Image-based insider threat detection via geometric transformation," *Security and Communication Networks*, vol. 2021, Article ID 1777536, 2021.

[8] G. Gavai, K. Sricharan, D. Gunning, R. Rolleston, J. Hanley, and M. Singhal, "Detecting insider threat from enterprise social and online activity data," in *Proceedings of the 7th ACM CCS International Workshop on Managing Insider Security Threats*, pp. 13–20, Colorado, DN, USA, October 2015.

[9] P. Chattopadhyay, L. Wang, and Y.-P. Tan, "Scenario-based insider threat detection from cyber activities," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 3, pp. 660–675, 2018.

[10] S. Yuan, P. Zheng, X. Wu, and Q. Li, "Insider threat detection via hierarchical neural temporal point processes," in *Proceedings of the 2019 IEEE International Conference on Big Data (Big Data)*, pp. 1343–1350, IEEE, Los Angeles, CA, USA, December 2019.

[11] T. Rashid, I. Agrafiotis, and J. R. Nurse, "A new take on detecting insider threats: exploring the use of hidden Markov models," in *Proceedings of the 8th ACM CCS International Workshop on Managing Insider Security Threats*, pp. 47–56, Vienna, Austria, October 2016.

[12] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson, "Deep learning for unsupervised insider threat detection in structured cybersecurity data streams," in *Proceedings of the Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*, San Francisco, March 2017.

[13] L. Liu, C. Chen, J. Zhang, O. De Vel, and Y. Xiang, "Insider threat identification using the simultaneous neural learning of multi-source logs," *IEEE Access*, vol. 7, pp. 183 162–183 176, 2019.

[14] J. Glasser and B. Lindauer, "Bridging the gap: a pragmatic approach to generating insider threat data," in *Proceedings of the 2013 IEEE Security and Privacy Workshops*, pp. 98–104, IEEE, San Francisco, CA, USA, May 2013.

[15] A. P. Moore, D. A. Mundie, and M. L. Collins, "A system dynamics model for investigating early detection of insider threat risk," in *Proceedings of the 31st International Conference of the System Dynamics Society*, Pittsburgh, PA, USA, July 2013.

[16] S. Wasko, R. E. Rhodes, M. Goforth et al., "Using alternate reality games to find a needle in a haystack: an approach for testing insider threat detection methods," *Computers & Security*, vol. 107, Article ID 102314, 2021.

[17] M. Collins, *Common Sense Guide to Mitigating Insider Threats*, Carnegie-Mellon Univ Pittsburgh Pa Pittsburgh United States, Pittsburgh, PA, USA, Tech. Rep, 2016.

[18] I. Homoliak, F. Toffalini, J. Guarnizo, Y. Elovici, and M. Ochoa, "Insight into insiders and it: a survey of insider threat taxonomies, analysis, modeling, and countermeasures," *ACM Computing Surveys*, vol. 52, no. 2, pp. 1–40, 2019.

[19] S. Yuan and X. Wu, "Deep Learning for Insider Threat Detection: Review, Challenges and Opportunities," *Computers & Security*, vol. 104, Article ID 102221, 2021.

[20] S. L. Pfleeger, J. B. Predd, J. Hunker, and C. Bulford, "Insiders behaving badly: addressing bad actors and their actions," *IEEE Transactions on Information Forensics and Security*, vol. 5, no. 1, pp. 169–179, 2009.

[21] J. R. Nurse, O. Buckley, P. A. Legg et al., "Understanding insider threat: a framework for characterising attacks," in *Proceedings of the 2014 IEEE Security and Privacy Workshops*, pp. 214–228, IEEE, San Jose, CA, USA, May 2014.

[22] D. C. Le and N. Zincir-Heywood, "Anomaly detection for insider threats using unsupervised ensembles," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1152–1164, 2021.

[23] P. Ferreira, D. C. Le, and N. Zincir-Heywood, "Exploring feature normalization and temporal information for machine learning based insider threat detection," in *Proceedings of the 2019 15th International Conference on Network and Service Management (CNSM)*, pp. 1–7, IEEE, Halifax, Canada, October 2019.

[24] D. C. Le and N. Zincir-Heywood, "Exploring adversarial properties of insider threat detection," in *Proceedings of the 2020 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, IEEE, Avignon, France, August 2020.

[25] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: a heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1777–1794, London, UK, November 2019.

[26] D. Sun, M. Liu, M. Li, Z. Shi, P. Liu, and X. Wang, "Deepmit: a novel malicious insider threat detection framework based on recurrent neural network," in *Proceedings of the 2021 IEEE 24th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*, pp. 335–341, IEEE, Dalian, China, May 2021.

[27] S. Yuan, P. Zheng, X. Wu, and H. Tong, "Few-shot insider threat detection," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2289–2292, Virtual Event, Ireland, 2020.

[28] A. Vaswani, N. Shazeer, N. Parmar et al., "Attention is all you need," in *Proceedings of the Advances in Neural Information Processing Systems*, pp. 5998–6008, Vancouver, Canada, 2017.

[29] D. C. Le, N. Zincir-Heywood, and M. I. Heywood, "Analyzing data granularity levels for insider threat detection using machine learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 1, pp. 30–44, 2020.

[30] T. E. Senator, H. G. Goldberg, A. Memory et al., "Detecting insider threats in a real corporate database of computer usage activity," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1393–1401, Chicago,USA, August 2013.

[31] K. Nance and R. Marty, "Identifying and visualizing the malicious insider threat using bipartite graphs," in *Proceedings of the 2011 44th Hawaii International Conference on System Sciences*, pp. 1–9, IEEE, Kauai, HI, USA, January 2011.

[32] L. Liu, C. Chen, J. Zhang, O. De Vel, and Y. Xiang, "Doc2vec-based insider threat detection through behaviour analysis of multi-source security logs," in *Proceedings of the 2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pp. 301–309, IEEE, Guangzhou, China, January 2020.

[33] D. Zhang, Y. Zheng, Y. Wen et al., "Role-based log analysis applying deep learning for insider threat detection," in *Proceedings of the 1st Workshop on Security-Oriented Designs of Computer Architectures and Processors*, pp. 18–20, Toronto, Canada, October 2018.

[34] H. Eldardiry, E. Bart, J. Liu, J. Hanley, B. Price, and O. Brdiczka, "Multi-domain information fusion for insider threat detection," in *Proceedings of the 2013 IEEE Security and Privacy Workshops*, pp. 45–51, IEEE, San Francisco, CA, USA, May 2013.

[35] A. Coden, W. Lin, K. Houck et al., "Uncovering insider threats from the digital footprints of individuals," *IBM Journal of Research and Development*, vol. 60, no. 4, pp. 8–1, 2016.

[36] P. A. Legg, O. Buckley, M. Goldsmith, and S. Creese, "Automated insider threat detection system using user and role-based profile assessment," *IEEE Systems Journal*, vol. 11, no. 2, pp. 503–512, 2015.

[37] L. Lin, S. Zhong, C. Jia, and K. Chen, "Insider threat detection based on deep belief network feature representation," in *Proceedings of the 2017 International Conference on Green Informatics (ICGI)*, pp. 54–59, IEEE, Fuzhou, China, August 2017.

[38] L. Liu, O. De Vel, C. Chen, J. Zhang, and Y. Xiang, "Anomaly-based insider threat detection using deep autoencoders," in *Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, pp. 39–48, IEEE, Singapore, November 2018.

[39] L. Liu, C. Chen, J. Zhang, O. De Vel, and Y. Xiang, "Unsupervised insider detection through neural feature learning and model optimisation," in *Proceedings of the 13th International Conference on Network and System Security*, pp. 18–36, Sapporo, Japan, December 2019.

[40] C. Zhou and R. C. Paffenroth, "Anomaly detection with robust deep autoencoders," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 665–674, Halifax, Canada, August 2017.

[41] B. Zong, Q. Song, M. R. Min et al., "Deep autoencoding Gaussian mixture model for unsupervised anomaly detection," in *Proceedings of the International Conference on Learning Representations*, Vancouver, Canada, February 2018.

[42] Q. P. Nguyen, K. W. Lim, D. M. Divakaran, K. H. Low, and M. C. Chan, "Gee: a gradient-based explainable variational autoencoder for network anomaly detection," in *Proceedings of the 2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 91–99, Washington, DC, USA, August 2019.

[43] D. Gong, L. Liu, V. Le et al., "Memorizing normality to detect anomaly: memory-augmented deep autoencoder for unsupervised anomaly detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1705–1714, Montreal, Canada, 2019.

[44] H. Park, J. Noh, and B. Ham, "Learning memory-guided normality for anomaly detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14360–14369, Seattle, WA, USA, March 2020.

[45] Y. Mirsky, T. Doitshman, Y. Elovici, and A. Shabtai, "Kitsune: an ensemble of autoencoders for online network intrusion detection," 2018, https://arxiv.org/abs/1802.09089.

[46] L.-P. Yuan, P. Liu, and S. Zhu, "Recompose event sequences vs. predict next events: a novel anomaly detection approach for discrete event logs," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pp. 336–348, Virtual Event, Hong Kong, May 2021.

[47] B. Zhao, L. Fei-Fei, and E. P. Xing, "Online detection of unusual events in videos via dynamic sparse coding," in *Proceedings of the CVPR 2011*, pp. 3313–3320, IEEE, Colorado Springs, CO, USA, June 2011.

[48] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures," in *Proceedings of the International Conference on Machine Learning. PMLR*, pp. 115–123, Edinburgh, Scotland, July 2013.

[49] P. Parveen and B. Thuraisingham, "Unsupervised incremental sequence learning for insider threat detection," in *Proceedings of the 2012 IEEE International Conference on Intelligence and Security Informatics*, pp. 141–143, IEEE, Washington, DC, USA, June 2012.

Hindawi
Security and Communication Networks
Volume 2022, Article ID 1289175, 16 pages
https://doi.org/10.1155/2022/1289175

Research Article

A Deep Learning Method for Android Application Classification Using Semantic Features

Zhiqiang Wang,[1,2,3] **Gefei Li**,[2] **Zihan Zhuo**,[4] **Xiaorui Ren**,[1] **Yuheng Lin**,[1] **and Jieming Gu**[4]

[1]*Department of Cyberspace Security, Beijing Electronic Science & Technology Institute, Beijing 100070, China*
[2]*State Information Center, Beijing 100045, China*
[3]*Guangdong Provincial Key Laboratory of Information Security Technology, Shenzhen, Guangdong 510006, China*
[4]*National Internet Emergency Center, Beijing, 100029, China*

Correspondence should be addressed to Zihan Zhuo; zzh@cert.org.cn

Received 8 December 2021; Accepted 1 February 2022; Published 24 February 2022

Academic Editor: Robertas Damaševičius

Copyright © 2022 Zhiqiang Wang et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Android has become the most popular mobile intelligent operating system with its open platform, diverse applications, and excellent user experience. However, at the same time, more and more attackers take Android as the primary target. The application store, which is the main download source for users, still does not have a complete security authentication mechanism. Given the above problems, we designed an Android application classification model based on multiple semantic features. Firstly, we use analysis tools to automatically extract the application's dynamic and static features into the text document and use variance and chi-square tests to optimize the features. Combined with natural language processing (NLP), we transform the feature file into a two-dimensional matrix and use the convolution neural network (CNN) to learn features efficiently. Also, to make the model satisfy more application scenarios, we design a dynamic adjustment method according to user requirements, the number of features, and other indicators. The experimental results demonstrate that the detection accuracy of malware is 99.3921%. We also measure this model's performance in detecting a malware family and benign application, with the classification accuracy of 99.5614% and 99.9046%, respectively.

1. Introduction

Android has many devices and users and rich applications as the most popular mobile intelligent operating system, bringing great convenience to people's lives. The open-source Android platform has made more and more mobile terminal manufacturers and developers join the Android alliance. According to the International Data Corporation's global smartphone market data report [1], with the popularization of 5G and the accelerated research and development of 5G smart terminals by notable brands, global smartphone shipments are expected to increase slightly by 1.6% next year. However, at the same time, the security problem of the Android system is also increasingly prominent, which contains more and more sensitive information such as user identity information, location information, and privacy data. At present, the security authentication mechanism for Android application stores is still not complete, and more and more attackers take the Android system as the primary attack target.

The Android platform provides some security mechanisms to limit malware functions, especially Android's permission control mechanism. Android system defines various permissions for developers to protect system resources and provides the corresponding APIs for accessing the above system resources. If an application wants to use these APIs to access user data, system configuration, and other resources, it must apply for the corresponding permissions and obtain the user's consent. However, most users usually blindly grant all permissions, thus destroying the

permission mechanism's effectiveness and failing to limit malware functions. As the primary source for users to download applications, various third-party application stores need to keep malware out and quickly and accurately classify benign applications automatically. Currently, application stores generally classify applications according to categories specified by developers or by analyzing descriptions provided by developers. However, malware developers can easily manipulate this process to evade detection, such as adding unqualified financial applications to the information application interface that is more easily approved. Additionally, as the number of applications explodes, it is becoming critical to classify applications quickly and accurately to improve management efficiency.

At present, most malicious application detection schemes use static or dynamic analysis methods to extract features and then combine machine learning algorithms to identify malicious applications. Some schemes directly visualize the application source code as a gray image or RGB image and use deep learning technology to analyze image features. Although the feature extraction step is omitted, it still belongs to static analysis. MalNet [2] uses CNN and LSTM networks to learn from the grayscale image and opcode sequence and takes a stacking ensemble for malware classification. Ganesh et al. [3] extracted 138 permission features and converted them into 12×12 PNG images and then used CNN to detect malicious applications. Xu et al. [4] used the control flow graph, data flow graph, and their possible combination as the Android application features, then encoded the graph into a matrix, and used them to train the classification model through CNN. Zegzhda et al. [5] proposed an approach for representing an Android application for a CNN, which consists of constructing an RGB image, the pixels of which are formed from a sequence of pairs of API calls and protection levels.

In the above detection method, CNN has two key benefits: local invariance and compositionality. Local invariance allows us to classify an image as a process containing a specific target, no matter where the target appears in the image. Compositionality means that each filter combines features to form a high-level representation, enabling the network to learn more precious features at a deeper level. However, the above methods are static analysis, which cannot wholly characterize the behavior of malicious applications, and need to extract dynamic features. Yuan et al. [6] proposed an online Android malware detection engine that extracted 192 features using static and dynamic analysis techniques and combined with Deep Belief Networks to detect malware, achieving high accuracy. However, there are still some problems, such as low accuracy and long training time for high dimensional feature detection. In NLP, Kim [7] proposed applying CNNs to sentence-level classification problems and achieved excellent results with an uncomplicated model.

This paper combines NLP and CNN to extract static and dynamic features and adds their frequency to describe features more accurately. Then, all the features contained in each application are transformed into a two-dimensional matrix. Finally, we use CNN to learn features efficiently to classify applications quickly. The main contributions of this paper are as follows:

(1) Automated feature extraction: We use four dynamic and static analysis tools to extract features and express the features in the form of "feature name + frequency" to describe the features more accurately and comprehensively. Each type of feature set is independent, and users can flexibly choose analysis tools according to their needs. This paper conveniently adds new feature set types to ensure the long-term validity of the model.

(2) Feature vector generation: We transform the feature file into a two-dimensional matrix using NLP and combine it with CNN, which is excellent in image recognition, to learn features efficiently.

(3) Dynamic adjustment of the model: We design a dynamic adjustment method of parameters according to the user's requirements (binary classification/multiclassification), the number of applications, and the average number of features contained in the feature files, to ensure that the model can always maintain the best detection effect as the applications and features change.

(4) Aiming at the problem of multiclassification of malware families and benign applications, we design a multiclassification method for Android applications based on CNN, which has higher detection accuracy than other methods.

2. Related Work

There are numerous Android malware detection schemes, mainly divided into static analysis methods and dynamic analysis methods. The static analysis method analyzes source code files or executable files without running applications.

EveDroid [8] is an event-aware Android malware detection system that exploits the behavioral patterns in different events to detect new malware based on the insight that events can reflect applications' possible running activities. Kumar et al. [9] proposed an Android malware detection framework based on machine learning and blockchain. Machine learning automatically extracts the malware information using clustering and classification techniques and storing it into the blockchain. Hasegawa and Iyatomi [10] proposed a light-weight Android malware detection method. It treats a minimal part of the target's raw APK file as a short string and analyzes it with one-dimensional CNN. Zhang et al. [11] proposed an Android malware detection method based on the method-level correlation relationship of the application's abstracted API calls. It calculates the confidence of association rules between the abstract API calls to form the behavioral semantics that describes applications and then build the detection system in combination with machine learning. Fang et al. [12] proposed an Android malware familial classification method based on DEX file section features. It first converts the DEX file into RGB image and plain text, respectively, and then extracts the image's and text's

color and texture as features. Finally, a feature fusion algorithm based on multiple kernel learning is used for classification. Apposcopy [13] is a new semantics-based approach for detecting Android malware. It incorporates a high-level language for specifying malware signatures and a static analysis for deciding if a given application matches a given signature. TaeGuen et al. [14] proposed using the method based on presence and similarity to extract features and using a multimodal deep learning method to detect malware. MADAM [15] is a host-based malware detection system for Android devices. It simultaneously analyzes and correlates features at four levels, kernel, application, user, and package, to detect and stop malicious behaviors. Narayanan et al. [16] proposed a method that uses control flow graphs as features and uses online support vector machine algorithms to detect malicious applications. Azad et al. [17] used particle swarm optimization to perform feature selection, a set of features to characterize the behavior of android applications and classify them as legitimate and malicious. Nisa et al. [18] proposed a feature fusion method that combines features extracted from pretrained AlexNet and Inception-v3 deep neural networks with features obtained from images representing malware code using segmentation-based fractal texture analysis (SFTA) and built a multimodal representation of malicious code for classifying grayscale images. Hemalatha et al. [19] describe malware binaries as 2D images and classify them with a deep learning model. Feng et al. [20] analyze and extract two types of features (i.e., manifest attributes and API calls) directly from the Dalvik binary and further update the feature input with matching results between text-based behavioral descriptions and code-level features.

The above static analysis methods have the advantages of fast detection speed and high efficiency and can detect malware in large quantities. The disadvantage is that they cannot fight against code transformation technology and dynamic malicious payload technology. The dynamic analysis method can overcome the above weakness [21]. It can capture sensitive behaviors in real time dynamically. Feng et al. [22] proposed EnDroid, which uses DroidBox to extract behavioral features through a runtime monitor and uses chi-square feature selection algorithms and ensemble learning to detect malware. Enck et al. [23] designed a Taint Droid detection tool, which marks a variety of sensitive data with taints. It determines whether the application has a privacy data leakage behavior by monitoring the flow path of these contaminated sensitive data in real time in a sandbox environment. Tam et al. [24] proposed a dynamic system based on a virtual machine called CopperDroid, directly detecting system calls to determine the operating system's actions, generating detailed and semantic behavior information to identify malicious applications. However, it can only identify the interaction between the system and the application, not the interaction between the applications. The above methods are not affected by the code transformation technology and can analyze the application's behavior in-depth, but the time is expensive. To analyze Android applications more comprehensively, we use dynamic and static analysis methods to extract features.

3. Architecture

The Android application classification model's overall architecture is shown in Figure 1. It is mainly divided into five modules: feature extraction module, feature preprocessing module, feature vector generation module, deep learning module, and detection module. First, we rename the collected applications with the file hash, remove the duplicate applications, and then store the applications' file hashes with the label "benign" or "malicious" in the database.

In the feature extraction module, we use static and dynamic analysis to batch extract features of applications into text documents. Each line represents a feature, and each application corresponds to a feature file.

In the feature preprocessing module, we use feature selection algorithms to optimize features further. Next, in the feature vector generation module, we convert each feature file into a two-dimensional matrix.

In the deep learning module, the model can flexibly adjust the parameters of the CNN according to the user's need and detection conditions (high precision/high efficiency), detection types (binary classification/multi-classification), the average number of features, and other indicators and select the most appropriate model as the final detection model through training.

Finally, the user submits the application to be tested through the client. The malware detection module firstly checks whether the application already exists in the database through file hash and, if so, directly returns the detection result. If it does not exist, the optimal detection model obtained by the deep learning module is used. Details of each module are shown in Figure 1.

3.1. Feature Extraction Module. In the feature extraction module, for each application to be tested, we use APKTool [25], androguard [26] Drozer, and DroidBox [21] to analyze the applications, obtain the corresponding files, and then extract features from them. The feature consists of two parts separated by spaces, the feature's name, and the frequency of the feature occurrence. It will be omitted if the frequency is 1. If the frequency is 0, this feature is not selected. For each tool, we separately write Python scripts to extract features into text documents automatically and then merge them into the final feature files for each application. The following introduces the four tools and the extracted content.

3.1.1. APKTool. We use APKTool to decompile the application to get AndroidManifest.xml. by parsing XML tree nodes <uses-permission>, <intent-filter>, <uses-feature> to extract permission features, component features, and environmental features.

(1) Permission Features. When an application performs specific operations or accesses certain data, it must apply for corresponding permissions, which means that the permissions defined in the manifest file can indicate the application's behavior. In the feature extraction process, we only extract the permission name. We collect the system

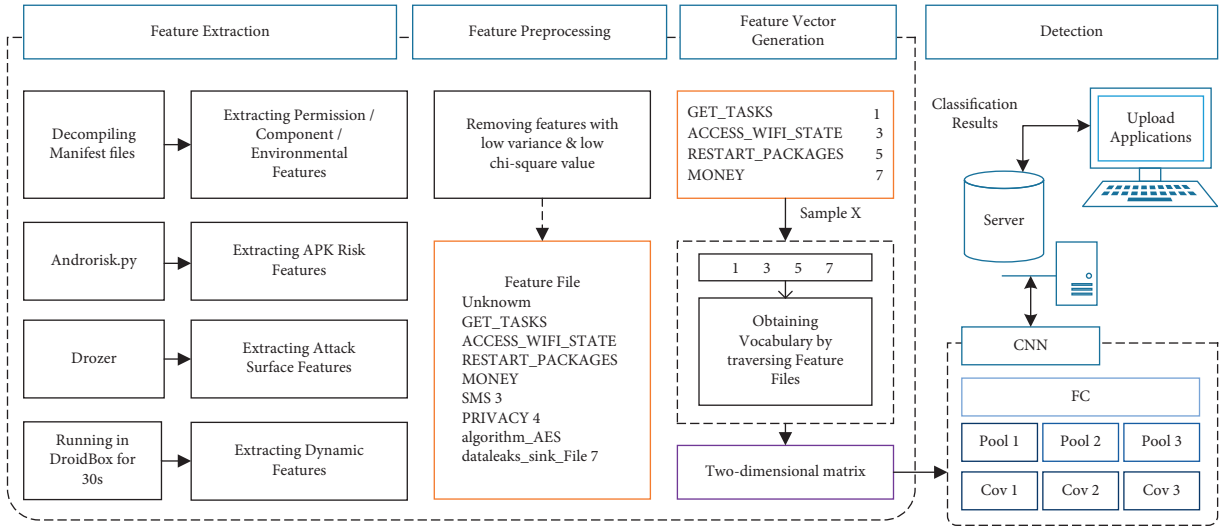


Figure 1: The overall architecture of the model.

permissions of the application by parsing the <uses-permission> tags.

(2) Component Features. Application components are the basic building blocks of Android applications, including Activity, Service, Broadcast Receiver, and Content Provider. Components are called Intents, which can register and receive messages. We can use them to start components or pass some important data to components. We collect component features by parsing the <action> and <category> tags in the <intent-filter> tags.

(3) Environmental Features. It includes hardware or software functions that applications depend on, such as GPS and NFC. Devices lacking specific hardware or software functions will not execute applications that require such special functions. For example, Android devices that do not support wireless charging cannot charge wirelessly. We collect environmental features by parsing <uses-feature> tags.

3.1.2. Androguard. We use the androrisk.py file in androguard to analyze the applications' risk level, and the analysis results and extracted contents are shown in Figure 2.

The analysis results are mainly composed of three parts: DEX, APK, and PERM. The analysis contents of DEX and APK are given in Table 1. PERM is the number of different functional permissions.

3.1.3. Drozer. Drozer is an Android security testing framework. We use "app. package. attacksurface" command to test the attackable points of the applications and extract the attack surface features. The results are shown in Figure 3.

3.1.4. DroidBox. TaintDroid is a dynamic stain detection technology, whose core idea is to mark sensitive data and turn them into pollution sources. In the process of program operation, when these pollution sources spread through interprocess communication, file transfer, etc., TaintDroid will conduct tracking reviews and record in the log to realize the tracking of sensitive data. DroidBox builds on this by performing dynamic stain analysis at the application framework level and redefines the types of stain tags, adding functions such as file manipulation monitoring, network sending and receiving data monitoring, encryption and decryption logging, and log analysis. DroidBox provides two scripts, startemu.sh for launching an emulator dedicated to dynamic analysis of Android apps and droidbox.sh for performing specific dynamic analysis tasks. This paper extracts dynamic behavior features from the operation logs of each application by running the DroidBox for 30 seconds. The specific features are as follows:

(1) Cryptographic Operation. Malicious applications usually use encryption to encrypt root vulnerabilities, malicious payloads, key method identifiers, value-added service SMS, and URLs to remote malicious servers to avoid static detection. So, we count the frequency of encryption, decryption, and key generation and record all encryption algorithms used by the applications.

(2) Network Operation. Malicious applications may receive messages from malicious command and control (C & C) servers through the network and obtain malicious payloads from malicious websites, so that attackers can manipulate the applications to obtain users' private information. We count the frequency of sending and receiving network communication data.

(3) Information Leaks. Information leaks are mainly through the network and files, so we count the number of times dataleaks_operation_write, dataleaks_sink_File, dataleaks_operation_read, dataleaks_-sink_Network occurred. Simultaneously, the leakage of LOCATION, IMSI, ICCID, IMEI, PHONE_NUMBER, LOCATION_GPS is also calculated.

The Analysis Results of Androrisk.py

/mnt/hgfs/share/apk/1.apk
RedFlags
DEX { 'NATIVE' :0, 'DYNAMIC' :1, 'CRYPTO' :1, 'REFLECTION' :1}
APK { 'DEX' :0, EXECTABLE' :0, 'ZIP' :0, 'SHELL_SCRIPT' :0, 'APK' :0, 'SHARED LIBRARIES' :3}
PERM { 'PRIVACY' :0, 'NORMAL' :1, 'MONEY' :0, 'INTERNET' :1, 'SMS' :0, 'DANGEROUS' :1, 'SIGNATUREORSYSTEM' :0, 'CALL' :0, 'SIGNATURE' :0, 'GPS' :0}
FuzzyRisk
VALUE 92.0

Feature File

DYNAMIC
CRYPTO
REFLECTION
SHARED LIBRARIES 3
NORMAL
INTERNET
DANGEROUS

Figure 2: The analysis result and extracted contents of androrisk.py.

Table 1: The analysis content of androrisk.py.

DEX	Description
NATIVE	Number of calls to non-java code
DYNAMIC	Times of dynamic loading of dex from sd
CRYPTO	Number of hidden dexes
REFLECTION	Number of reflections
APK	Description
DEX	Times of dex use
EXECUTABLE	Number of executions
ZIP	Compressed package
SHELL_SCRIPT	Number of times the script is used
APK	Number of other apks
SHARED LIBRARIES	Number of shared databases

```
dz> run app. package. attacksurface com. glu. android. dinercn
Attack Surface:
  1 activities exported
  1 broadcast receivers exported
  0 content providers exported
  0 services exported
dz>
```

Figure 3: The results of attack surface test.

(4) Sent SMS. Malicious applications usually cause financial charges to infected users. They can secretly subscribe to value-added services by sending several SMS messages without the user's consent. So, we count the frequency of sending text messages.

(5) Service Start. Malicious applications usually perform malicious behavior in background processing contained in in-service components. So, we count the number of times the service has started.

(6) Receiver Action. Malicious applications usually leverage system events to trigger malicious behaviors. Registered Broadcast Receivers can be a fair reflection of the monitored system events. For example, registering the reception of BOOT_COMPLETED intent in malware indicates triggering malicious activity directly after the mobile device's startup.

3.2. Feature Preprocessing Module. To further reduce overfitting and improve the model's training speed and generalization ability, we design a feature preprocessing module. The first step is to remove low-variance features. Through experimental tests, setting the removal rate to 99.95% and above can achieve better detection results. The second step is the chi-square test, which can express the correlation between feature items and categories. The higher the CHI value, the more significant the correlation. So, we remove the features with lower chi-square values in the experiment.

3.3. Feature Vector Generation Module. According to reference [27], after the feature preprocessing module, we traverse all the feature files and obtain all the features that have appeared as the vocabulary. Each feature in the vocabulary is labeled with consecutive numbers to obtain a mapping from feature to label ID. Also, we add an "Unknown" feature to match unknown features that are not in the vocabulary during the detection phase.

Since CNN's input is a vector in continuous space, while NLP uses discrete characters, we need to use word embedding technology to convert each feature file into a two-dimensional matrix when classifying Android applications using CNN. First, we represent each feature in the vocabulary with a vector and randomly initialize the vectors. Then, we update the word vector continuously with training. The length of the word vector depends on the specific situation of the feature set. 50–300 is a common choice, and we set it to 200. We convert the features contained in each feature file into corresponding ID sequences according to the vocabulary, respectively. Then, the feature file is transformed into a two-dimensional matrix according to the ID sequences and the vocabulary. The specific process is shown in Figure 4.

3.4. Deep Learning Module. Deep learning algorithms include CNN, RNN, and LSTM. LSTM and RNN are suitable for learning long time series, and CNN has a better learning ability for local features. In this paper, the features contained in each application are composed of four parts, which have no time sequence and short average length. According to reference [6], the parts associated with permission features can more effectively characterize applications' malice. That is, capturing the relationship between local features can train the model more effectively. So, we finally adopted CNN.

The structure of CNN in the deep learning module is shown in Figure 5.

The first layer is the embedding layer, which is mainly responsible for embedding features into low-dimensional vectors. Then, we perform multiple parallel convolution

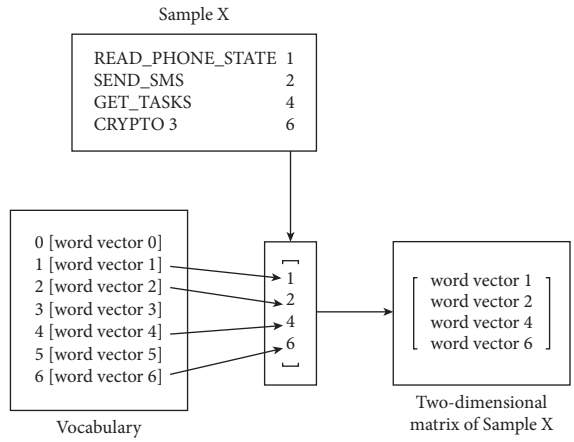


Figure 4: Feature vector generation.

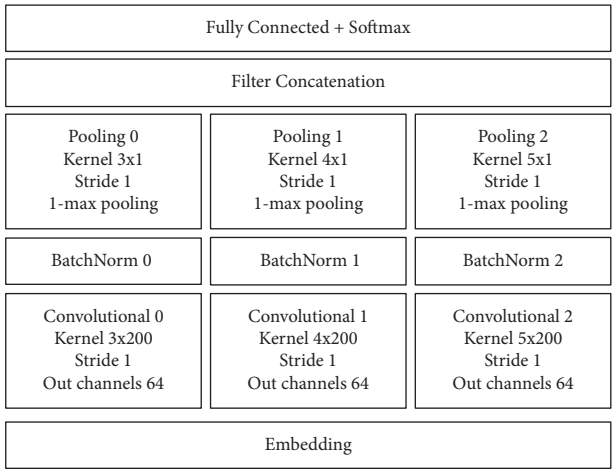


Figure 5: The structure of CNN.

operations, batch normalization, and 1-max pooling on the input matrix and concatenate all the outputs into a fixed-length feature vector. Finally, we classify the results using the full connection layer. The specific description of each layer is as follows:

3.4.1. Convolutional Layer. The convolutional layer is the core of the network. We use multiple filters of different sizes to learn the same area's complementary features, and we can obtain different feature maps. There are three parallel convolutional layers in Figure 5, where the width of the filter is the same as the width of the two-dimensional matrix (i.e., the length of the word vector), and we set it to 200. In this way, after a convolution operation, the two-dimensional matrix becomes a column vector. (3,4,5) is the height of the filter, that is, the relationship between 3, 4, and 5 features. There are 64 filters of each type.

3.4.2. Pooling Layer. The dimension of the feature map generated by each filter varies depending on the number of features and the size of the filter region. Therefore, 1-max pooling is applied to each feature map to induce a fixed-length vector.

3.4.3. Fully Connected Layer. We send the concatenate vectors to the softmax classifier through the full connection layer for classification and use the regularization technology Dropout to prevent overfitting.

3.5. Detection Module. The user submits the application to be tested through the client. The malware detection module firstly checks whether the application already exists in the database through file hash and, if so, directly returns the detection result. If it does not exist, the optimal detection model obtained by the deep learning module is used. The detection result is an excel file, including the file hash and classification results of the application (0 for malicious application and 1 for benign application).

4. Experiments and Evaluation

4.1. Dataset. The dataset includes 18549 malware and 18453 benign applications, of which the malware comes from VirusShare [28] and Drebin [29], and the benign applications come from Google Play Store [30]. They are scanned and detected by VirusTotal [31]. When all virus scanners in VirusTotal treat the application as benign, the application will be included in the benign application dataset.

In practice, we analyzed all applications, but tools do not correctly analyze some applications. So, the features are contained in 100% of the manifest files, 99.6757% of the APK risk files, 99.5081% of the attack surface files, and 72.6420% of the dynamic behavior files. Detailed statistical results are shown in Table 2.

4.2. Experimental Settings. The experimental environment is as follows.

(i) Hardware Dependencies: on the hardware, NVIDIA GPU GeForce RTX 2070, and 8 GB memory are used.

(ii) Software Dependencies: in terms of software, Ubuntu 16.04 LTS, Python 3.6, TensorFlow 1.13.1, Scikitlearn 0.20.3, Numpy 1.16.2, Pandas 0.24.2, and Matplotlib 3.0.3 are used.

(iii) GPU Components: GPU components include NVIDIA GPU driver, CUDA 10.1 and cuDNN v7.5.1.

GPU is used to accelerate the CNN. Tensorflow is used to implement CNN, and Scikitlearn is used to implement various machine learning algorithms.

Android malware detection is a binary classification problem. There are four possible prediction results. The confusion matrix is shown in Table 3. Among them, True Negative (TN) indicates that benign samples are predicted as benign, False Negative (FN) indicates that malicious samples are predicted as benign, False Position (FP) indicates that benign samples are predicted as malicious, and True Position (TP)

Table 2: Detailed statistics of dataset.

	Manifest file	APK risk file	Attack surface file	Dynamic behavior file
Malicious	18549 (100%)	18506 (99.7682%)	18498 (99.7251%)	16769 (90.4038%)
Benign	18453 (100%)	18316 (99.2576%)	18322 (99.2901%)	10110 (54.7878%)
Total	37002 (100%)	36882 (99.6757%)	36820 (99.5081%)	26879 (72.6420%)

Table 3: Confusion matrix.

Type of prediction	Benign	Malicious
Benign	TN	FP
Malicious	FN	TP

indicates that malicious samples are predicted as malicious. The evaluation indicators are as follows: Accuracy (ACC), Precision (PRE), Recall (REC), and F1 score (F1).

Accuracy is defined as the percentage of the total sample that is predicted correctly.

$$\mathrm{ACC} = \frac{(\mathrm{TP} + \mathrm{TN})}{(\mathrm{TP} + \mathrm{EN} + \mathrm{FP} + \mathrm{TN})}. \quad (1)$$

Precision means the probability of a positive sample among all the samples that are predicted to be positive.

$$\mathrm{PRE} = \frac{\mathrm{TP}}{(\mathrm{FP} + \mathrm{TP})}. \quad (2)$$

Recall is the probability of being predicted as a positive sample in a sample that is positive.

$$\mathrm{REC} = \frac{\mathrm{TP}}{(\mathrm{EN} + \mathrm{TP})}. \quad (3)$$

F1 score is the harmonized average of precision and recall.

$$F1 = 2\mathrm{PRE} \cdot \mathrm{REC}\,(\mathrm{PRE} + \mathrm{REC}). \quad (4)$$

4.3. Feature Selection and Analysis. After the feature extraction module, the number of features included in different feature sets is shown in Table 4 (the statistical results in this table only include the feature names and do not include the following parameters, such that "NATIVE 2" and "NATIVE 3" are counted only once in this table).

To have a more detailed understanding of the features and lay the foundation for subsequent experiments, after the feature preprocessing module, we count the types of features contained in each feature set (Type), and the maximum (Max), minimum (Min), and average (Ave) number of features included in all feature files of each feature set. The statistical information is shown in Table 5.

Taking feature set I as an example, we compare the statistical information before and after feature preprocessing and find that the number of feature types and the maximum number of features are significantly reduced. Personality features are greatly reduced, but the reduction of the average number of features is minimal.

4.4. Contrast Experiment with Machine Learning Algorithms. To analyze the effect of the Android malware detection model based on CNN, we select seven machine learning algorithms, namely, k-NearestNeighbor (kNN), Decision Tree (DT), Support Vector Machine (SVM), Logistics Regression (LR), XGBoost, Random Forest (RF), and Multi-Layer Perceptron (MLP) to test the detection effect on three feature sets.

The hyperparameters of seven machine learning algorithms are set by default. The hyperparameters of CNN mainly refer to the experimental results and parameter adjustment suggestions in reference [32]. Through a series of experimental verification, we have obtained the baseline configuration parameters.

The filter region size is (3,4,5), the feature map is 64, and the activation function is RELU. The embedding size and batch size are 200 and 128, respectively. The epoch is set to 10, 1-max pooling is used, and the dropout rate and train set radio are 0.5 and 50%, respectively.

We randomly selected 50% in all applications as the training set, of which 10% as the verification set and the remaining 50% as the test set. The experimental results are shown in Figure 6. The detection effect of CNN is better than that of machine learning algorithms. With the increase of features, the detection effect is getting better and better, proving the effectiveness of the feature set.

4.5. Contrast Experiment between Different Hyperparameters of CNN. This section further analyzes the influence of different hyperparameters on the experimental results. For this reason, keep all other settings unchanged and only change the parameters to be analyzed.

4.5.1. Effect of Filter Region Size. We first perform a coarse linesearch over a single filter region size to find the "best" size for the feature set under consideration. The experimental results are shown in Figure 7. When the filter region size is 1, the performance of the model is weak. When the filter region sizes are 3 and 5, the classification accuracy is the highest. According to the statistical information, the average number of features in each feature file ranges from 13 to 24. For feature files containing more features, the optimal filter region size can be more extensive. When the training set accounts for 50%, the size of the vocabulary corresponding to feature set I is 1233, feature set I + II is 1438, and feature set I + II + III is 1817.

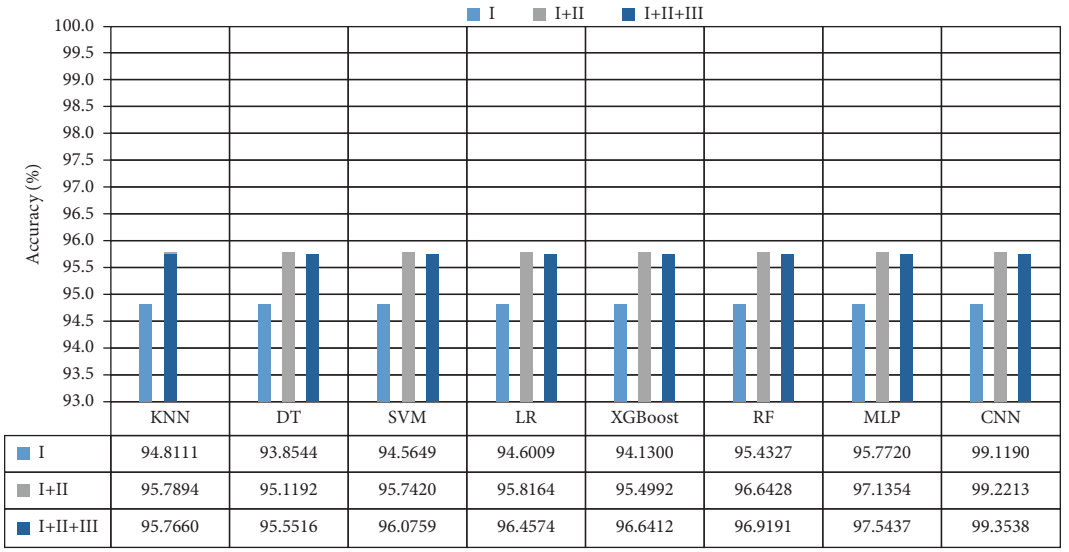
Besides, we combine different region sizes and copies to obtain the best effect. The experimental results can be seen in Table 6. Using different feature sets, the combination of several region sizes near the optimal size can improve the classification performance. However, when we use other

Table 4: The information of feature sets.

		Benign	Malicious
Feature set I	Permission feature	571	276
	Component feature	848	420
	Environmental feature	151	38
	Total	**1570**	**734**
Feature set II	APK risk feature	20	20
Feature set III	Dynamic feature	45	45
	Attack surface feature	4	4

Table 5: The statistical information of feature files.

		Type	Ave	Max	Min
I (before)	Malicious	734	16.4317	126	1
	Benign	1570	13.1863	193	1
I (after)	Malicious	286	16.3737	111	1
	Benign	352	13.0542	149	1
I + II (after)	Malicious	306	23.4436	121	1
	Benign	372	20.0836	164	1
I + II + III (after)	Malicious	355	24.3196	121	1
	Benign	421	20.4258	164	1



	KNN	DT	SVM	LR	XGBoost	RF	MLP	CNN
I	94.8111	93.8544	94.5649	94.6009	94.1300	95.4327	95.7720	99.1190
I+II	95.7894	95.1192	95.7420	95.8164	95.4992	96.6428	97.1354	99.2213
I+II+III	95.7660	95.5516	96.0759	96.4574	96.6412	96.9191	97.5437	99.3538

Figure 6: Detection accuracy of different algorithms under different feature sets.

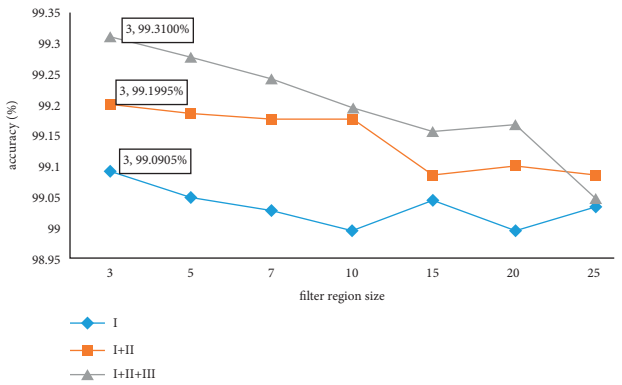


Figure 7: Accuracy of models with different filter region size.

Table 6: Results obtained using different feature sets.

Region size	Train time (s)	ACC (%)	PRE (%)	REC (%)	F1 (%)
Feature set I					
(3)	37	99.0905	98.9735	99.1794	99.0763
(5)	39	99.0508	98.3917	99.6995	99.0413
(3,4,5)	63	99.1190	99.0872	99.1216	99.1044
(2,3,4,5)	80	99.0962	98.7040	99.4683	99.0847
(3,3,3)	62	99.0280	99.2337	98.7864	99.0096
(3,3,3,3)	74	99.1303	98.7719	99.4683	99.1189
(7,7,7)	81	99.0621	98.9616	99.1331	99.0473
Feature set I + II					
(3)	36	99.1995	99.0590	99.3307	99.1947
(5)	38	99.1886	98.7705	99.6050	99.1860
(3,4,5)	63	99.2213	99.2966	99.1332	99.2148
(2,3,4,5)	76	99.2430	99.1781	99.2978	99.2379
(3,3,3)	58	99.1886	99.2096	99.1551	99.1824
(3,3,3,3)	71	99.2267	98.9524	99.4953	99.2231
Feature set I + II + III					
(3)	43	99.3100	98.8877	99.7360	99.3100
(5)	44	99.2771	99.0153	99.5381	99.2760
(3,4,5)	70	99.3538	99.0061	99.7030	99.3534
(2,3,4,5)	85	99.2552	99.0794	99.4281	99.2534
(3,3,3)	69	99.2662	99.0044	99.5271	99.2650
(3,3,3,3)	79	99.3045	98.9089	99.7030	99.3044

filter combinations far from the optimal region size, we cannot achieve a better detection effect than a single filter. For example, when using Feature Set I, the detection effect of a single filter (3) is better than that of filter combinations (7,7,7).

When all the features are applied, the model performs best when the filter region size is (3,4,5). The classification accuracy is 99.3538%, and the training time is 70 seconds. The change of accuracy and loss on the training set and verification set is shown in Figures 8 and 9. The orange line results from the training set, and the blue line is the result of the verification set. The model tends to be stable after 1,250 batch iterations, with the verification set's accuracy floating around 99.4% and the loss value floating around 0.02.

4.5.2. Effect of Number of Feature Maps. We again hold other configurations constant and test the influence of the number of feature maps. The experimental results are shown in Figure 10. The "best" number of feature maps depends on the feature sets. As the number of feature maps increases, the classification effect is getting better and better. However, when it exceeds the critical value marked in Figure 10, the model may reduce the classification accuracy due to overfitting. On the other hand, as the number of feature maps increases, the model's training time is longer.

4.5.3. Effect of Regularization. Dropout is a common regularization strategy. During the learning process of a neural network, it temporarily discards some units with a certain probability and discards the weights of all nodes connected to them. Keeping the other settings unchanged, we use the feature set I + II + III to test the effect of Dropout. The experimental result is shown in Figure 11. When the number of feature maps is 64, the model performs best when the dropout rate is 0.5. The classification accuracy is 99.3538%, and the training time is 70 seconds. At this time, the number of randomly generated network structures is also the largest. When the number of feature maps is 500, the model performs best when the dropout rate is 0.6. The classification accuracy is 99.3921%, and the training time is 200 s. By comparing the two sets of data, we can find that Dropout's effect is more noticeable when the amount of data is large. When the network has the risk of overfitting, we can try the following methods to prevent overfitting: 1. Applying batch standardization between convolution layers can regularize and avoid the gradient disappearance and reduce training time. 2. When Dropout is applied to the full connection layer, the dropout rate can be set to about 0.5.3, combining learning rate decay and Adam optimization algorithm to improve the model's detection effect further.

4.6. Contrast Experiment with Deep Learning Algorithms. To show the performance of our model, we investigated similar approaches that have been previously proposed. Table 7 shows the results of the investigations. Many existing methods utilize the malware samples from the VirusShare. Therefore, we include the performance in the table when using the samples from the VirusShare in the detection test.

As shown in Table 7, our model's detection accuracy and the F1-score values are higher than the other methods. [14, 33] adopt the method of generating feature vectors based on existence. If there is a corresponding feature in the application, it is expressed as 1; if it does not exist, it is expressed as 0. The vectors generated by this method are too sparse, and the dimension of feature vectors is high. Our method can overcome these shortcomings and achieve better detection results.

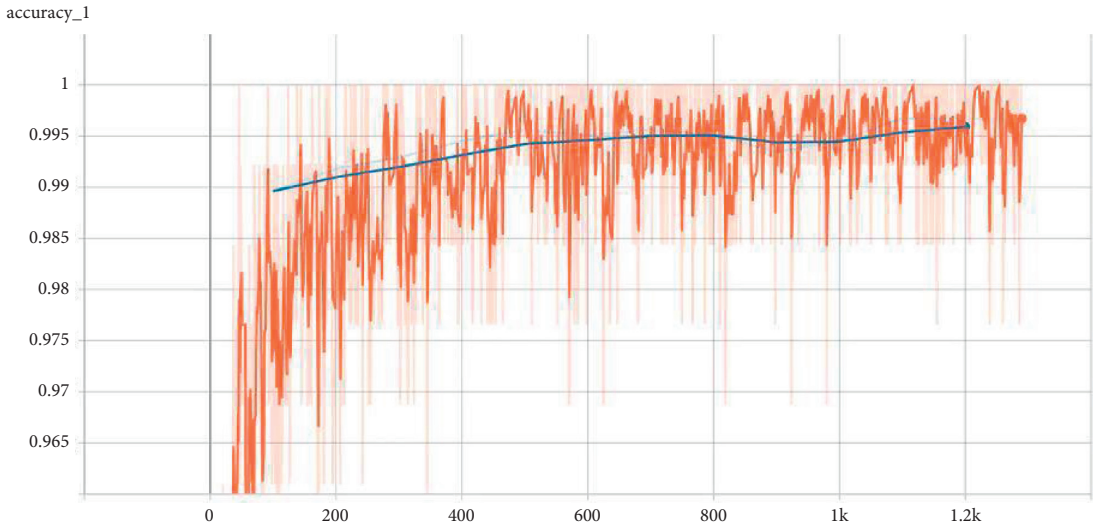


Figure 8: The change of accuracy.

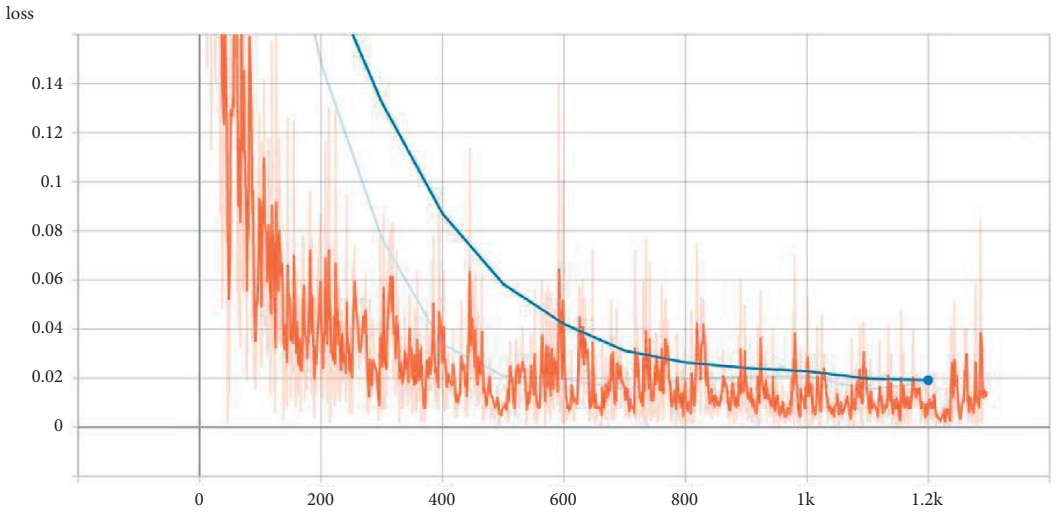


Figure 9: The change of loss.

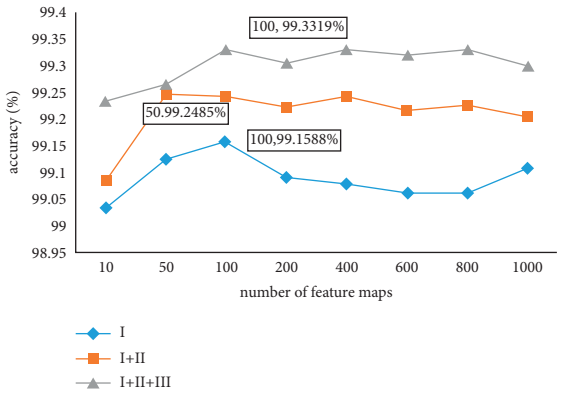


Figure 10: Effect of number of feature maps for different feature sets.

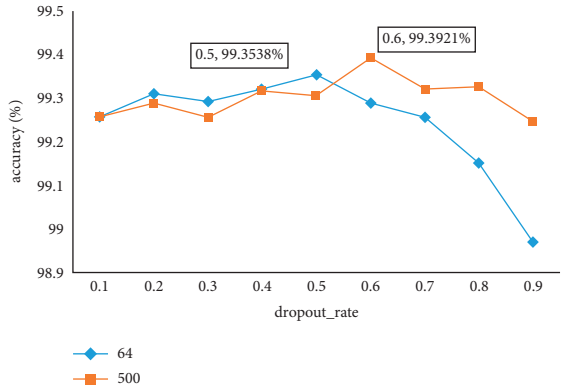


Figure 11: Effect of regularization.

Table 7: The results of the investigations.

System	Dataset		Algorithm	ACC/F1
	Benign applications	Malware		
Ours	Google play store: 18453	VirusShare + drebin: 18549	CNN	99.3538%/99.3534%
[33]	Google play store: 3000	Android malware genome project + Virusshare: 8000	DBN	NA/95.05%
[14]	Google play store: 19747	VirusShare: 13075 malgenome project: 1209	DNN	98%/NA
[34]	360 security company and wandoujia app store:10948	Drebin + VirusShare: 8652	CNN	96.54%/95.89%
[35]	AndroZoo: 5215	VirusShare + android malware genome project: 5442	DBN	98.71%/NA
[36]	Google play store: 8000	VirusShare: 8000	CNN	95.8%/NA

We measured the method proposed in [36], and the result is shown in Table 8. The feature vectorization method of [5] first reads the classes.dex file as an unsigned vector and then converts the vector into a fixed size by resampling. Resampling algorithms commonly used in image processing include Nearest Neighbor Interpolation, Bilinear Interpolation, and Bicubic Interpolation. Unlike two-dimensional images, [5] uses a similar method to resample one-dimensional sequences and convert the original bytecode of classes.dex into a fixed-size sequence. Besides, our model uses various kinds of features to reflect the various aspects of applications.

4.7. Dynamic Adjustment Method. To make the model meet more application scenarios, we design a dynamic adjustment method of the model according to the user requirements, the number of applications, and the average number of features. The detailed description is shown in Figure 12.

4.7.1. Mode Selection. The model includes three detection modes, namely, Android malware detection, Android malware familial classification, and benign application classification. Users can select the corresponding mode according to their own needs.

4.7.2. Training Set. We will regularly update and add the training set to ensure the model's adaptability and long-term effectiveness for Android malware detection mode. For Android malware familial classification mode, users need to upload the applications according to the familial classification or add/delete the familial applications based on Drebin. For benign application classification mode, users need to upload applications according to the application store's classification.

4.7.3. Tool Selection. Users can choose the detection tool according to their own needs (high efficiency/high accuracy), but to ensure the detection effect, APKTool is a required tool. Throughout the previous experiment, we observed that even if only feature set I is used, the model can still maintain a high accuracy of 99.1303% while detecting rapidly.

4.7.4. Obtaining Indicators. According to the tools that are selected by the user, feature extraction and preprocessing operations are performed to obtain indicators: the total number of feature files and the average number of features.

4.7.5. Dynamic Generation of Optimal Model. For multiclassification mode, users need to select detection tools and upload training sets. According to the indicators and conversion table shown in Table 9, CNN, with different parameters, is automatically generated to train the data. The model with the highest detection accuracy is stored as the final detection model.

Table 8: Comparison of experimental results.

	PRE (%)	REC (%)	ACC (%)	F1 (%)
Ours	99.0061	99.7030	99.3538	99.3534
[36]	95.4	96.2	95.8	95.8

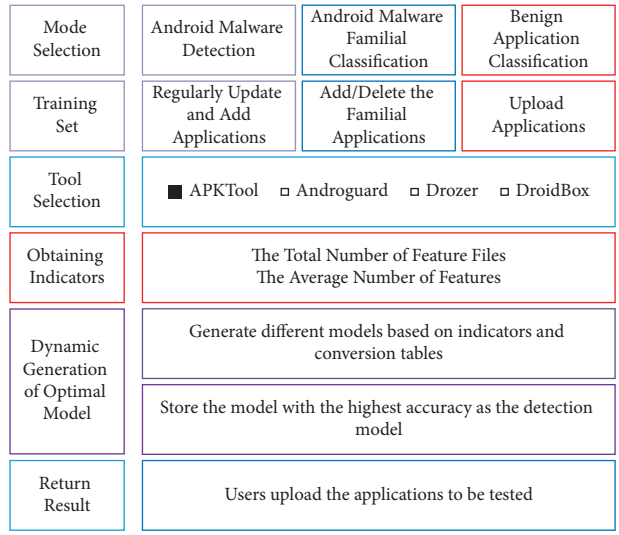


Figure 12: Dynamic adjustment method.

4.7.6. Return Result. The user uploads the applications to the server, and the server uses the detection tools selected by the user to perform feature extraction and preprocessing operations. Then, use the optimal mode generated in the previous step for detection, get the detection result, and return it to the user.

5. Detection of Malware Families

This section uses the Drebin dataset to evaluate the model's performance on the classification of malware families. The structure of CNN is shown in Figure 13, and the configuration parameters are as follows. The filter area size is set to (2, 3, 4, 5), the feature map is 16, and the activation function is RELU. The embedding size and batch size are 200 and 32, respectively, the epoch is set to 10, and 1-max pooling is selected for pooling. The dropout rate is 0.5, and the train set radio is 90%.

All malware belongs to known malware families. The basic information is shown in Table 10, including the number of applications of each malware family, the number of feature types included in feature set I, and feature set I + II.

In all applications, 90% are randomly selected as the training set, of which 10% are used as the verification set, and the remaining 10% are used as the test set. When the filter region size is (2,3,4,5), the model performs optimally, and the classification accuracy is 99.5614%. Two applications are classified incorrectly, which is higher than the 94.5% classification accuracy of EnDroid [22]. With the addition of feature set II, the classification accuracy and stability of the model have improved. The change of accuracy and loss on the training set and verification set are shown in Figures 14 and 15. The orange line is the result of the training set, and the blue line is the result of the verification set. The model tends to be stable after 1040 batch iterations, with the verification set's accuracy floating around 99% and the loss value floating around 0.04.

6. Detection of Benign Applications

This section evaluates the performance of the model on multiple classifications of benign applications. We collect nine types of applications from the Xiaomi App Store. The basic information is shown in Table 11, including the number of applications of each type (Apps), the types of features they contain (Features), and the maximum (Max), minimum (Min), and average (Ave) number of features contained in the feature files. From Table 11 [27], we can infer that the game applications do not have a specific function because the game applications contain only 191 types of features, and the average number of game applications' features is the smallest. While the sports applications features' minimum number is 11, which means they have specific functions. For example, accurately tracking sports routes, distances, speeds, and altitudes through GPS and measuring heartbeat frequencies and pulses related to medical sports are the typical functions in most sports applications. These applications need to get relevant permissions, namely, positioning permissions (ACCESS_FINE_LOCATION and ACCESS_COARSE_ LOCATION) and sensor permissions (BODY_SENSORS). It is feasible to implement classification according to the applications' features. Besides, to ensure the efficiency of classification management, we only use feature set I.

Table 9: Conversion table.

Number of features files	Number of filters	The average number of features	Filter region size
0–5000	16	10–50	(3,4,5) (3,3,3,3) (2,3,4,5)
5001–10000	32	51–100	(5,6,7) (5,5,5,5) (4,5,6,7)
10001–20000	64	100+	Step1: Search for a single filter region
20001–30000	128		Step2: Search for the combination of filters

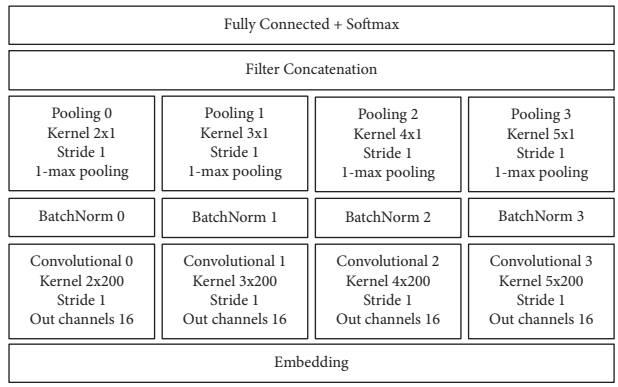


Figure 13: The structure of the CNN.

Table 10: The basic information of malware families.

Malware families	Samples	Feature set I	Feature set I + II
FakeInstaller	925	57	76
DroidKungFu	666	116	135
Plankon	625	133	152
Opfake	613	45	64
GinMaster	339	114	133
BaseBridge	328	66	85
Iconosys	152	33	52
$K_{\min}$	147	30	49
FakeDoc	132	49	68
Geinimi	91	43	62
Adrd	91	105	124
DroidDream	81	77	96
ExploitLinuxLotoor	69	67	86
MobileTx	69	10	29
Glodream	69	41	60
FakeRun	61	24	43
SendPay	59	18	37
Gappusin	58	46	65
Imlog	43	11	30
SMSreg	40	34	53
TOTAL	4658		

In the most relevant work, Shabtai et al. [37] use machine learning algorithms to classify tool and game applications by extracting static features from dex files and manifest files. Wang et al. [38] achieve an accuracy of 82.93% in benign application classification by extracting 11 static features and using a collection of multiple machine learning classifiers. Based on API relationships analysis and CNN, an automatic classification method for Android applications is proposed by Fan et al. [39]. It classifies applications into 24 categories with an average accuracy of 88.9%.

The configuration parameters of CNN are as follows, and its structure is the same as the malicious Android application familial classification model. The pooling and activation function select 1-max pooling and ReLU, respectively, the filter area size is (2,3,4,5), and the feature map is 32. The embedding size and batch size are 200 and 32, respectively, the epoch is set to 10, the dropout rate is set to 0.5, the train set radio is 80%, and the vocabulary is 1288. In all applications, 80% are randomly selected as the training set, of which 10% are used as the validation set, and the remaining 20% are used as the test set.

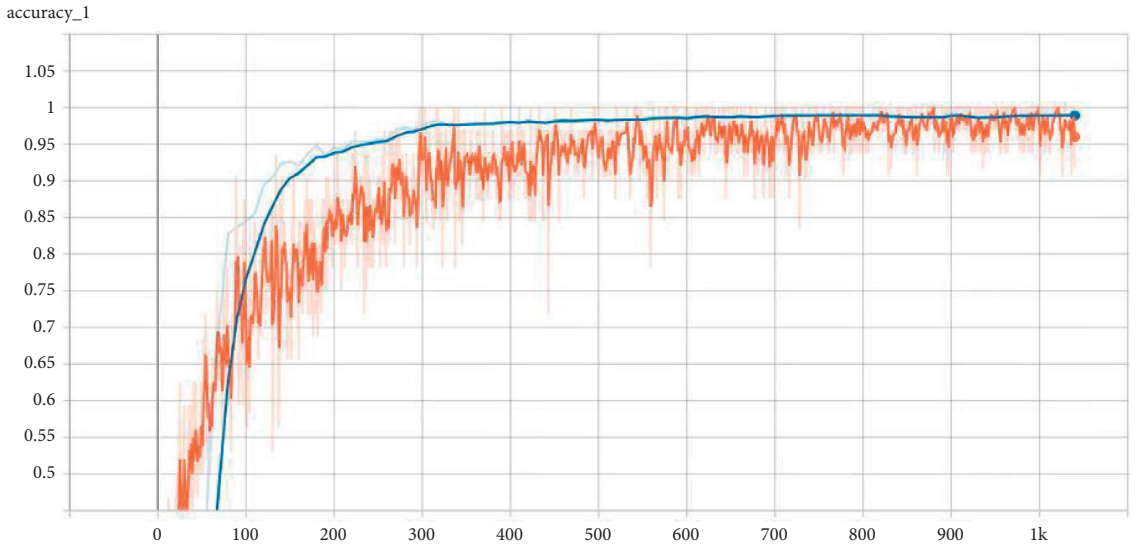


Figure 14: The change of accuracy.

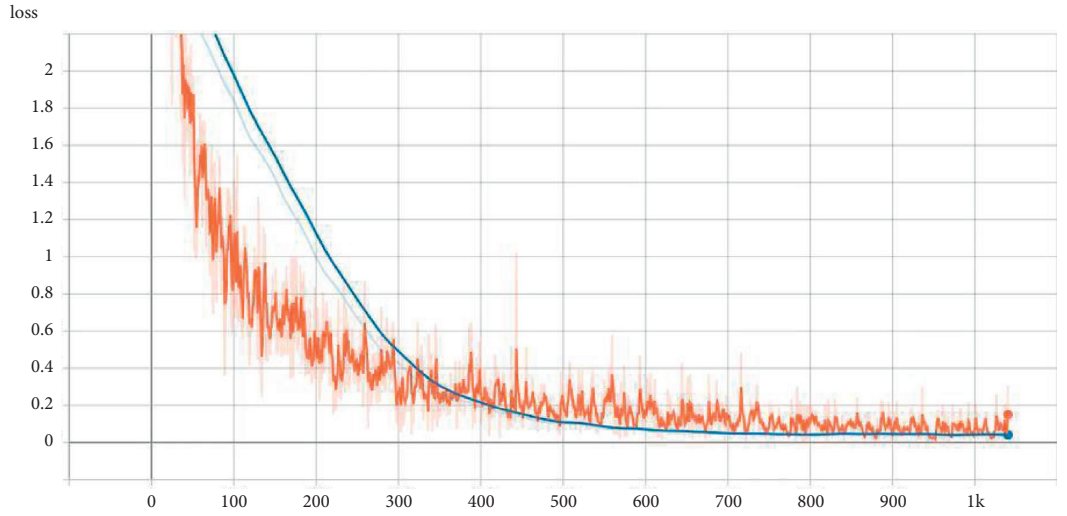


Figure 15: The change of loss.

Table 11: The basic information of feature files.

Type	Apps	Features	Max	Min	Ave
Game	697	191	59	1	13.4562
Book reading	597	450	121	4	33.5343
Audiovisual	659	520	98	3	34.8574
Chat social	741	484	161	1	39.4507
Sports	212	299	111	11	37.9057
News	552	381	122	2	37.5163
Shopping	803	567	132	1	39.7298
Financial	763	405	89	2	40.1782
Camera	323	289	107	1	23.2601

The experimental results are shown in Table 12. After 193s, the accuracy rate reaches 99.9046%. Only one application is misclassified. Suda, which is originally a chat social application, is misclassified as a camera application. After analysis, we found that Suda is a comprehensive application. It includes functions of chat social, taking pictures, and editing pictures simultaneously.

Table 12: Classification accuracy of benign application.

Region size	Train time (s)	ACC (%)
(3)	101	99.4275
(5)	103	99.5229
(3,4,5)	164	99.6183
(2,3,4,5)	193	99.9046
(3,3,3)	163	99.4275
(3,3,3,3)	188	99.6183

7. Conclusion and Future Work

In this paper, we design an Android application classification model based on multiple semantic features. It can extract multiple types of static and dynamic features automatically. We use feature selection algorithms to remove irrelevant or noisy features and extract critical features. These key features help identify dangerous behaviors in unknown applications more effectively. Then, we use CNN to implement classification. We verify the model's effectiveness, the usefulness of the feature sets, and the feature vector generation method's effect through a series of experiments. The model also performs well on malware familial classification and benign application classification and has a short training time.

Despite the effectiveness of our model, several issues remain to be resolved. Our future work will focus on addressing the following problems. During the dynamic analysis process, only 72.6420% of the applications can be correctly analyzed by DroidBox, and 45 useful features are extracted. We would investigate to combine input generator tools IntelliDroid [40] to improve dynamic analysis coverage and extract dynamic features in more detail. Finally, we will further refine our classification model to enable more accurate malware detection.

Data Availability

The data used to support the findings of this study have been deposited at https://github.com/blackwall0321/malicious_applications_detection.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was financially supported by China Postdoctoral Science Foundation funded project (2019M650606), the Opening Project of Guangdong Provincial Key Laboratory of Information Security Technology (202B1212060078-12), First-class Discipline Construction Project of Beijing Electronic Science and Technology Institute (3201012), and the National Key Research and Development Program of China under Grant (2018YFB0803401).

References

[1] International Data Corporation, "Global smartphone market data report [EB/OL]," 2020, https://www.idc.com/getdoc.jsp?containerId=prCHC45975020.

[2] J. Yan, Y. Qi, and Q. Rao, "Detecting malware with an ensemble method based on deep neural network," *Security and Communication Networks*, vol. 2018, no. 1, pp. 1–16, 2018.

[3] M. Ganesh, P. Pednekar, P. Prabhuswamy, D. S. Nair, Y. Park, and H. Jeon, "CNN-based Android malware detection," in *Proceedings of the 2017 International Conference on Software Security and Assurance (ICSSA)*, pp. 60–65, IEEE, Altoona, PA, USA, July 2017.

[4] Z. Xu, K. Ren, S. Qin, and F. Craciun, "CDGDroid: android malware detection based on deep learning using CFG and DFG," in *Proceedings of the International Conference on Formal Engineering Methods*, Springer, Cham, pp. 177–193, 2018.

[5] P. Zegzhda, D. Zegzhda, E. Pavlenko, and G. Ignatev, "Applying deep learning techniques for Android malware detection," in *Proceedings of the 11th International Conference on Security of Information and Networks*, vol. 7, September 2018.

[6] Z. Yuan, Y. Lu, and Y. Xue, "Droiddetector: android malware characterization and detection using deep learning," *Tsinghua Science and Technology*, vol. 21, no. 1, pp. 114–123, 2016.

[7] Y. Kim, "Convolutional neural networks for sentence classification," arXiv:1408.5882, 2014.

[8] T. Lei, Z. Qin, Z. Wang, Q. Li, and D. Ye, "EveDroid: event-aware android malware detection against model degrading for IoT devices," *IEEE Internet of Things Journal*, vol. 6, no. 4, pp. 6668–6680, 2019.

[9] R. Kumar, X. Zhang, W. Wang, R. U. Khan, J. Kumar, and A. Sharif, "A multimodal malware detection technique for android IoT devices using various features," *IEEE Access*, vol. 7, pp. 64411–64430, 2019.

[10] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for Android malware detection," in *Proceedings of the 2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, March 2018.

[11] H. Zhang, S. Luo, Y. Zhang, and L. Pan, "An efficient android malware detection system based on method-level behavioral semantic analysis," *IEEE Access*, vol. 7, no. 7, pp. 69246–69256, 2019.

[12] Y. Fang, Y. Gao, F. Jing, and L. Zhang, "Android malware familial classification based on DEX file section features," *IEEE Access*, vol. 8, no. 8, pp. 10614–10627, 2020.

[13] Y. Feng, I. Dillig, S. Anand, and A. Aiken, "Apposcopy: automated detection of Android malware (invited talk)," in *Proceedings of the 2nd International Workshop on Software Development Lifecycle for Mobile*, pp. 13-14, Hong Kong, China, November 2014.

[14] K. TaeGuen, K. BooJoong, R. Mina, S. Sezer, and E. G. Im, "A multimodal deep learning method for android malware detection using various features," *IEEE Transactions on Information Forensics and Security*, vol. 14, pp. 773–788, 2019.

[15] A. Saracino, D. Sgandurra, G. Dini, and F. Martinelli, "MADAM: effective and efficient behavior-based android malware detection and prevention," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 1, pp. 83–97, 2016.

[16] A. Narayanan, L. Yang, L. Chen, and L. Jinliang, "Adaptive and scalable android malware detection through online learning," in *Proceedings of the International Joint Conference*

on Neural Networks, pp. 2484–2491, Vancouver, BC, Canada, July 2016.

[17] M. A. Azad, F. Riaz, A. Aftab, S. K. J. Rizvi, J. Arshad, and H. F. Atlam, "DEEPSEL: a novel feature selection for early identification of malware in mobile applications," *Future Generation Computer Systems*, vol. 129, pp. 54–63, 2022.

[18] M. Nisa, J. H. Shah, S. Kanwal et al., "Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features," *Applied Sciences*, vol. 10, no. 14, p. 4966, 2020.

[19] J. Hemalatha, S. A. Roseline, S. Geetha, S. Kadry, and R. Damaševičius, "An efficient DenseNet-based deep learning model for malware detection," *Entropy*, vol. 23, no. 3, p. 344, 2021.

[20] R. Feng, S. Chen, X. Xie, G. Meng, S.-W. Lin, and Y. Liu, "A performance-sensitive malware detection system using deep learning on mobile devices," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1563–1578, 2021.

[21] P. Chaurasia, "Dynamic analysis of Android malware using droidbox," Dissertations & Theses, Gradworks, 2015.

[22] P. Feng, J. Ma, C. Sun, X. Xu, and Y. Ma, "A novel dynamic android malware detection system with ensemble learning," *IEEE Access*, vol. 6, no. 6, pp. 30996–31011, 2018.

[23] W. Enck, P. Gilbert, S. Han et al., "TaintDroid," *ACM Transactions on Computer Systems*, vol. 32, no. 2, pp. 1–29, 2014.

[24] K. Tam, S. J. Khan, A. Fattori, and L. Cavallaro, "CopperDroid: automatic reconstruction of android malware behaviors," in *Proceedings of the Internet Society Network and Distributed System Security Symposium*, pp. 15–26, San Diego, California, February 2015.

[25] "APKtool," 2019, https://ibotpeaches.github.io/Apktool.

[26] A. Desnos, G. Gueguen, and S. Bachmann, "Androguard package [EB/OL]," https://androguard.readthedocs.io/en/latest/api/androguard.html.

[27] Z. Wang, G. Li, and Y. Chi, "Multi-classification of android applications based on convolutional neural networks," in *Proceedings of the CSAE 2020: The 4th International Conference on Computer Science and Application Engineering*, Sanya, China, October 2020.

[28] "VirusShare," 2019, https://virusshare.com.

[29] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: effective and explainable detection of android malware in your pocket," in *Proceedings of the 2014 Network and Distributed System Security Symposium*, San Diego, California, February 2014.

[30] Google Play Store, https://play.google.com/store, 2019.

[31] VirusTotal, https://www.virustotal.com/ko[Online], 2019.

[32] Y. Zhang and B. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," 2015, https://arxiv.org/abs/1510.03820.

[33] D. Zhu, H. Jin, Y. Yang, D. Wu, and W. Chen, "DeepFlow: deep learning-based malware detection by mining Android application for abnormal usage of sensitive data," in *Proceedings of the 2017 IEEE symposium on computers and communications (ISCC)*, pp. 438–443, IEEE, Heraklion, July 2017.

[34] D. Zhu, T. Xi, P. Jing, D. Wu, Q. Xia, and Y. Zhang, "A transparent and multimodal malware detection method for android apps," in *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pp. 51–60, FL, Miami Beach, USA, November 2019.

[35] X. Qin, F. Zeng, and Y. Zhang, "MSNdroid: the Android malware detector based on multi-class features and deep belief network," in *Proceedings of the ACM Turing Celebration Conference-China*, pp. 1–5, Chengdu, China, May 2019.

[36] Z. Ren, H. Wu, Q. Ning, I. Hussain, and B. Chen, "End-to-end malware detection for android IoT devices using deep learning," *Ad Hoc Networks*, vol. 101, p. 102098, 2020.

[37] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated static code analysis for classifying android applications using machine learning," in *Proceedings of the International Conference on Computational Intelligence and Security*, pp. 329–333, Nanning, China, December 2010.

[38] W. Wang, Y. Li, X. Wang, J. Liu, and X. Zhang, "Detecting Android malicious apps and categorizing benign apps with ensemble of classifiers," *Future Generation Computer Systems*, vol. 78, pp. 987–994, 2018.

[39] W. Fan, Y. Chen, Y. A. Liu, and F. Wu, "DroidARA: android application automatic categorization based on API relationship analysis," *IEEE Access*, vol. 7, pp. 157987–157996, 2019.

[40] M. Y. Wong and D. Lie, "IntelliDroid: a targeted input generator for the dynamic analysis of android malware," in *Proceedings of the Network & Distributed System Security Symposium*, Ontario, Canada, January 2016.